Private Cloud Management

Transforming VMware SDDC into Your IaaS



PART 4 OF 4

September 2026

Iwan "e1" Rahabok
VMware Cloud Foundation | Broadcom
e1@broadcom.com

5 Edition

Back of cover page. Delete if you do not plan to print.
Thank you Mai Mai for the design. Daddy proud of you!

PART 4

Miscellaneous

Table of Contents

This part is the place for a variety of stuff. Some of them may grow into their own chapters or parts in the future. It also covers basic knowledge that maybe useful for those without computer science background.

Chapters:

Super Metrics	Co-authored by Peter Tymbel.
Automation	By Thomas Kopton
SDDC vs IaaS	
Epilogue	Personal sharing from me about the life of an infrastructure architect

Super Metrics

Part 4 Chapter 1

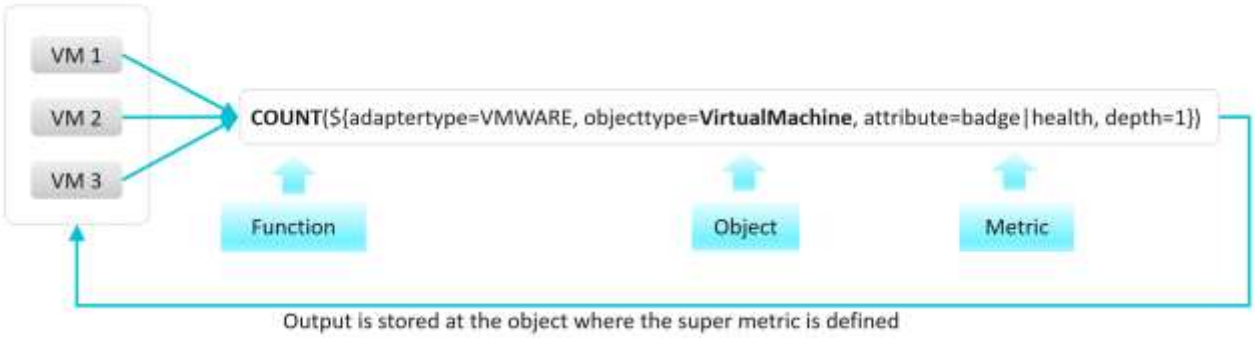
I've used the product since 1.0. When I first saw super metrics, it was one of those "is this created for me?" moments. The ability to analyze hundreds of metrics by creating my own metrics enabled me to do my job better. I was a pre-sales engineer but I spent a lot of time helping my customers troubleshooting and optimizing their environment.

Introduction

Super metrics are derived metrics that can be used to measure performance, utilization, and cost by different business units and applications, or in this case, the Custom Group into which the VMs are placed. Super metrics contain simple algebraic formulas that allow Aria Operations to measure various aspects of the Custom Groups, which contain business unit–related VMs.

At the most basic level, super metrics enable the ability to extend built-in metrics by adding new metrics that provide additional insight. This is especially important for Custom Groups as they only come with basic badge and population metrics. Having a set of more comprehensive predefined metrics is not optimal since Custom Groups can contain a mix of vastly different objects such Hosts and Datastores. For example: CPU related metrics would not be applicable to the Datastores.

This is also where the not-so-obvious power of super metrics becomes apparent. The super metrics employed to quantify Custom Groups use the concept of Relative Reference, or Depth, which allows Aria Operations to look inside of a container/bucket, measure what is in it, and return the value at the container level rather than the object being measured. This ability empowers the user to create a number of algorithms leveraging the built-in functions and collected time-series data.



It is a programming language with mathematical formula that contains a combination of one or more metrics for one or more objects. Once you create a super metric, it persists over time and can be used for different interactions just like any other metric. It can be used in views, reports, dashboards and symptom definitions.

Super metrics are created with a list of operators and functions. These can be further combined and operated using several conditional expressions like 'where' or 'if-else'.

Note that it calculates the present value. You can't compare against historical data. That means you can't do things such as counting the number of times an object crosses a particular threshold. You can only set the value to 1 if the threshold is crossed. To see the total over time, use the summation feature in View List.

Before starting to create a super metric, make sure you identify the following:

- Objects or object types that are involved
- Metrics which will need to be used.
- How to combine the metrics? Which operator, function or expression to use?
- Which object type will be used to assign super metric?
- Policy in which super metric will need to be enabled

Basic Functions

The maximum, minimum, average and sum functions are simple functions that can quickly summarize a large number of objects. I will provide some examples demonstrating the various features of super metrics functions and operators. More is covered in the manual.

There are also many examples in the repository of super metrics.

Example 1

Maximum CPU Ready (%) among VMs within a group of clusters providing same class of service.

```
max( ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|readyPct, depth=3 } )
```

Maximum Memory Balloon (%) among VMs within a group of clusters providing same class of service.

```
max( ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=mem|balloonPct, depth=3 } )
```

Depth=3 is used since the super metric is applied at a custom group level which is 3 levels up the VM object whose metric is used. The hierarchy is Group → Cluster → ESXi Host → VM.

Tip: If you use the same super metric for different levels, specify the deepest one.

Example 2

Sum of vCPUs provisioned on all VMs in a group of VM.

```
sum( ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|corecount_provisioned, depth=1 }
)
```

Depth=1 is sufficient as the VM is directly under the group. No need to manually change the depth.

Average of CPU Usage with all VMs in a custom group:

```
avg( ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|usage_average, depth=3 } )
```

Default Value

Return CO2 Emission if the metric exists. If not, return 0.744 by default.

```
${this, metric=CustomProperty|CO2 Emission, defval=0.744}
```

The above is handy if the metric may not exist and you want to specify a default value.

'Where' Clause

Things get more powerful and complex once you need to specify a condition.

Use Case: Count of all VMs in the environment which has CPU usage greater than 60% at that time.

```
count(
     ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|usage_average, depth=5,
        where=($value > 60)
      }
     )
```

Note: you specify 60 not 60% or 0.6. It has to match the metric value.

Use Case: Count of all Microsoft Windows VMs.

That means you need to do a string comparison. You also need to know the actual values used by the property field.

```
count(
     ${ adaptertype=VMWARE, objecttype=VirtualMachine, attribute=summary|guest|fullName, depth=5,
        where="summary|guest|fullName startsWith 'Microsoft Windows'"
      }
     )
```

Another example

```
sum(
    ${ adaptertype=VMWARE, objecttype=HostSystem, attribute=mem|host_provisioned, depth=1,
       where="runtime|connectionState equals connected"
     }
   )
```

The choice for string comparison is

```
equals
contains
startsWith
endsWith
!equals
!contains
!startsWith
!endsWith
```

Use Case: Compute the percentage of VMs with CPU Ready > 1%.

That means you need to divide the number of VM against the total number of running VM.

```
count(
     ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|readyPct, depth=8, where=">1" }
     )
/
${ this, metric=summary|running_vms }
* 100
```

The last line in the code is to manually convert into percentage.

Use Case: Count of all VMs with CPU usage > 70% OR memory usage > 60%

This is a double comparison, with an OR clause. The formula gets complex as super metric is actually a run time code that gets executed directly. There is no translation!

```
count(
      ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=cpu|usage_average, depth=8,
         where= ( $value > 70
                  ||
                  ${metric=mem|usage_average } > 60
                )
       }
    )
```

Notice the first comparison simply uses the variable $value, because it's actually defined in the metric=.

Use Case: Count of all VMs which are not Windows based or Redhat based.

This means you need to negate the comparison. The negation has to be done outside the two comparison.

```
count(
     ${ adaptertype=VMWARE, objecttype=VirtualMachine, metric=summary|guest|fullName, depth=5,
        where= (! ($value contains 'Microsoft Windows' || $value contains 'Redhat') )
      }
     )
```

If Then Else

This takes you deeper into Java programming 😊

Use Case: Count of provisioned vCPUs and if it is equal to 4, return value "1" and if it is not equal to 4, return a value "0".

```
count( ${this, metric=cpu|corecount_provisioned, depth=1, where= ($value == 4)} )
? 1
: 0
```

Use Case: Find the "Actual Recommended vCPU" for a VM.

While using the rightsizing feature, VCF Operations provide the vCPUs to be removed or added based on if it is an oversized or undersized VM. The following logic can be used find the actual recommended values:

If the value of Recommended vCPUs to add is 0,

then Actual Recommended vCPU = Provisioned vCPUs – Recommended vCPUs to remove (as an Oversized VM),

else Actual Recommended vCPU = Provisioned vCPUs + Recommended vCPUs to add (as an Undersized VM).

```
${this, metric=summary|undersized|vcpus} == 0
?
( ${this, metric=cpu|corecount_provisioned} - ${this, metric=summary|oversized|vcpus} )
:
( ${this, metric=cpu|corecount_provisioned} + ${this, metric=summary|undersized|vcpus} )
```

This formula uses the $this, a reference to the object itself. So the context is not other object.

For completeness, let's do the same for memory. Since the default unit is KB, we would like to convert into GB.

If the value of Recommended Memory to add is 0,

then Actual Recommended Memory = Provisioned memory – Recommended Memory to remove (as an Oversized VM),

else Actual Recommended Memory = Provisioned Memory + Recommended Memory to add (as an Undersized VM)

```
(
  ${this, metric=summary|undersized|memory} == 0
  ?
  ( ${this, metric=mem|guest_provisioned} - ${this, metric=summary|oversized|memory} )
  :
  ( ${this, metric=mem|guest_provisioned} + ${this, metric=summary|undersized|memory} )
)
/ 1048576
```

Take note there is no Case Statement. So you gotta use Nested IF as workaround.

Instanced Value

Some metrics and properties in VCF Operations exist as instance value, meaning they are repeated for each instance. The advantage of this it reduces the number of children object. The disadvantage is in visualization, as the metrics need to be accessed via the instance and not just the object.



One way to make reporting or dashboarding easier is to aggregate the values to the object. You do this via the attribute option.

The following code adds all virtual disks space of a VM.

```
Sum (${adaptertype=VMWARE, objecttype=VirtualMachine, attribute=virtualDisk|configuredGB,
depth=5})
```

Note that where clause can be done with metric only and not attribute. You can add instanced metrics but cannot apply where clause to that.

Advanced Examples

This is where you vrealize the full capability of super metric. It's actually a programming language that gets executed as a straight line. That means it can't loop. Think of it as a mathematical formula.

RDM Disk Capacity

How do you sum the total capacity of RDM disks in a virtual machine, since the VM typically has VMDK disk also?

In this super metric, we check the virtual disk for the property "IsRDM" and if it is true, add that disk capacity, else set the capacity value as zero. We do it for each scsciN:N, and sum all the answers.

Limitation: you need to keep adding the disks as required.

```
(sum
   (${this, metric=virtualDisk:scsi0:0|configuredGB, where= "virtualDisk:scsi0:0|isRDM equals
true"}) ?
   (${this, metric=virtualDisk:scsi0:0|configuredGB}) : 0
)
+
(sum
   (${this, metric=virtualDisk:scsi0:1|configuredGB, where= "virtualDisk:scsi0:1|isRDM equals
true"}) ?
   (${this, metric=virtualDisk:scsi0:1|configuredGB}) : 0
)
+
(sum
   (${this, metric=virtualDisk:scsi0:2|configuredGB, where= "virtualDisk:scsi0:2|isRDM equals
true"}) ?
   (${this, metric=virtualDisk:scsi0:2|configuredGB}) : 0
)
```



Tools Status

Use Case: check the VM Tools running status. If it is running, return the value of OS uptime, else return the value zero.

In this example, we're combining where clause and If Then Else.

```
count( ${this, metric=summary|guest|toolsRunningStatus, where= (!($value contains 'Not
Running'))}) != 0
?
( ${this, metric=sys|osUptime_latest} )
```

```
:
0
```

Weighted KPI

Use Case: define Performance (%) of an object. The requirement is defined in the Performance Modelling section of vSphere Metrics book.

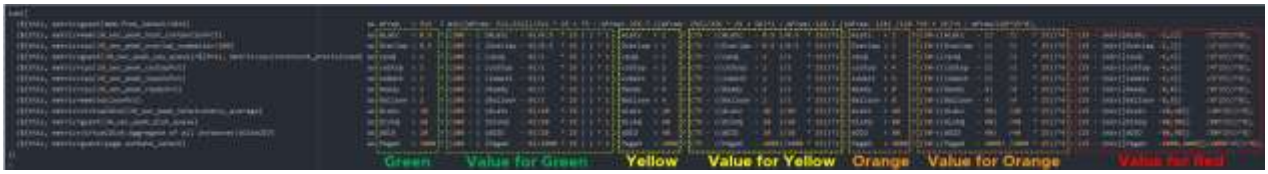
Let's say you have the following metrics that you want to combine into a single KPI

RAM Page-in Rate (MB/s)	0 – 25	25 - 50	50 – 100	100 – 200
CPU Co-Stop (%)	0 – 1%	1 – 2	2 – 3	3 - 4
CPU Ready (%)	0 – 2.5%	2.5 – 5	5 – 7.5	7.5 - 10
CPU IO Wait (%)	0 – 1%	1% - 3%	3% - 5%	5% - 7%

Take Other Wait for example.

- The green range is only from 0% - 1%
- That means 0% Other Wait equals to 100%, while 1% Other Wait translates into 75%.
- That means 0.8% Other Wait is 80%.
- The yellow range is 2x wider than the green range. 2% Other Wait gets translated into 62.5% as it's in between 50% and 75%.

The overall code maps to the table above. The following code snippet shows the blocks for green, yellow, orange and red. No need to zoom this picture, as we will dive into the actual code line by line.



Summary

Super metric does not have a Case statement, so we have to use nested IF. The logic looks something like this

```
If   it's in the green range
then          calculate for green range
else if it's in the yellow range
     then     calculate for yellow range
     else if in the orange range
          then calculate for orange range
          else calculate for red range
```

In addition to the above, you also need to assign weightage. This is critical if you have many metrics forming a KPI. for example, if there are 10 metrics, then a single red will not have enough weight to change the overall KPI red. To solve this, you assign the following weightage:

- yellow 2x the weight of green

- Orange 4x the weight of green
- Red 8x the weight of green.

Let's see first part of the code. I'm using CPU Other Wait as the example:

```
(${this, metric=cpu|iowaitAvg} as ioWait < 1
?         (100 - ( (ioWait - 0)           / 1 * 25 ) ) * 1
: ioWait < 2
    ?     ( 75 - ( (ioWait - 1)           / 1 * 25 ) ) * 2
    : ioWait < 4
        ? ( 50 - ( (ioWait - 2)           / 2 * 25 ) ) * 4
        : ( 25 - ( min([ ioWait - 4 ,4 ]) / 4 * 25 ) ) * 8
```

Let's step through it.

The code "as ioWait" defines an alias. Don't think of Alias as variable. All it does is giving a short or friendly name to a metric. They don't work for expressions.

We need to do the following translation:

IO Wait	0%	1%	2%	4%	8% or more
Translation	100	75	50	25	0

The values in between need to follow the range of each bracket. Notice the width of green bracket is only 1% but the yellow is 2%. The red is special as anything above 7% needs to be capped to 0.

Using the above, let's show a few examples of translation:

- 0.2% Other Wait → 95.0
- 0.5% Other Wait → 87.5
- 2.0% IO Wait → 67.5
- 3.4% IO Wait → 45.0

Green Range

In a perfect score, the logic will basically do 100 – 0, hence returning 100%.

In a non-perfect score, we need to prorate the value, so they land somewhere between 75 and 100.

Now let's see how the translation is done for the Other Wait metric. The `(1-0)` is the range of green.

As the range is going downward, meaning the higher the IO Wait, the lower the result, it's more intuitive to deduct from 100 than to add from 75.

If the IO Wait < 1 it performs the following

```
100  - ( (ioWait - 0) / (1 - 0) ) * 25
```

Assuming IO Wait = 0.2%, we get

```
100  - ( ( 0.2 - 0)   /    1    ) * 25
100  - (           0.2           ) * 25
100  -                5
```

You get 95, which is correct.

Yellow Range

The yellow has 2x weightage of green. So we need to multiply the final value by 2x.

```
( ioWait < 3 ? ( 75 - ((ioWait - 1) / (3 - 1) * 25)  *  2
```

The line `: ioWait < 3` is the ELSE branch of IF IO Wait < 1%. So we just need to check for < 3% and can safely assume it's larger than 1%.

The `(3-1)` is the range of yellow. We can replace it with just the number 2 for more efficient code.

```
( 75 - ((ioWait - 1) / 2  * 25) )  *  2
```

Assuming IO Wait = 1.4%, we get

```
( 75 - ( (1.4  - 1)  / 2  * 25) )  *  2
( 75 - (     0.4     / 2  * 25) )  *  2
( 75 -               5          )  *  2
              70                    *  2
```

You get 140, which is correct.

The constant 50 is required to bump up the value as yellow starts at 50.

Notice you don't need to do that funky flipping anymore 😊

Orange Range

The same logic applies for orange. The only difference is you Orange range is 50 - 25. You then multiply by 4 as it's 4x green.

The `(5-3)` is the range of yellow. We can replace it with just the number 2 for more efficient code.

```
( 50 - ((ioWait - 3) / (5-3)  * 25) )   *    4
   50 -     3 - 3    / 2     * 25
   50        0 / 2 * 25
  50 - 0
```

Red Range

The red range is 5 – 7%. So 5% translates into 25%, while anything 7% or more becomes 0%.

As red is at the end of the range, we need to deal with corner case by capping any value above 7% and return 0.

The portion `min([ (ioWait - 5),2 ])` caps the value to maximum 2 when IO Wait is 7% or more.

For IO Wait just above 5%, we want to return value near 2, so we get larger multiplier.

```
( 25 - ( min([ ioWait - 5 ,2 ]) / (7 - 5) * 25 )  )    * 8
```

Assuming IO Wait = 5.4%, we get

```
( 25 - ( min([  5.4   - 5 ,2 ]) /     2    * 25)  )    * 8
( 25 - ( min([     0.4    ,2 ]) /     2    * 25)  )    * 8
( 25 - (         0.4           /     2    * 25)  )    * 8
( 25 -                  5                         )    * 8
                        20                              * 8
```

You get 160, which is correct.

Weightage

Can you spot a missing logic in the above?

Yes, the multiplier creates a problem. When you multiply by 2x, 4x, 8x, you need to normalize it back to the values fall within 0 – 100%.

To normalize the value back to 100%, you need to divide by the multiplier.

But how do you normalize, since each metric can have their own multiplier?

You need to have another set of nested IF statement, this time you increase the denominator correspondingly. The following shows the logic for 2 of the metrics. The multiplier is shown in red.

```
Sum ([
    ( ioWait < 1 ? 1 : ( ioWait < 2 ? 2 : ( ioWait < 4 ? 4 : 8 ) ) ) ,
    ( Co-stop < 1 ? 1 : ( Co-stop < 2 ? 2 : ( Co-stop < 4 ? 4 : 8 ) ) )
])
```

Once you have the above 2 sets, it's a matter of dividing one over the other. The following shows part of the logic, as I want to focus on the 2 sum statements.

```
sum([
  (${this, metric=cpu|20_sec_peak_iowaitPct} as ioWait < 1 ? (100 - ( (ioWait - 0)/1 * 25 ) ) * 1
  (${this, metric=cpu|20_sec_peak_readyPct}  as Ready  < 2 ? (100 - ( (Ready  - 0)/2 * 25 ) ) * 1
   ])
/
sum([
    (ioWait  < 1    ? 1 : (ioWait  <   2 ? 2 : (ioWait  <    4 ? 4 : 8))) ,
    (Ready   < 2    ? 1 : (Ready   <   4 ? 2 : (Ready   <    8 ? 4 : 8))) ,
   ])
```

The complete multiplier part of the code looks like this

```
sum([
    (mFree   < 128  ? 8 : (mFree   < 256 ? 4 : (mFree   <  512 ? 2 : 1))) ,
    (mLatc   < 0.5  ? 1 : (mLatc   <   1 ? 2 : (mLatc   <    2 ? 4 : 8))) ,
    (Overlap < 0.5  ? 1 : (Overlap <   1 ? 2 : (Overlap <    2 ? 4 : 8))) ,
    (cpuQ    < 1    ? 1 : (cpuQ    <   2 ? 2 : (cpuQ    <    4 ? 4 : 8))) ,
    (coStop  < 1    ? 1 : (coStop  <   2 ? 2 : (coStop  <    4 ? 4 : 8))) ,
    (ioWait  < 1    ? 1 : (ioWait  <   2 ? 2 : (ioWait  <    4 ? 4 : 8))) ,
    (Ready   < 2    ? 1 : (Ready   <   4 ? 2 : (Ready   <    8 ? 4 : 8))) ,
    (Balloon < 2    ? 1 : (Balloon <   4 ? 2 : (Balloon <    8 ? 4 : 8))) ,
    (dLatc   < 10   ? 1 : (dLatc   <  20 ? 2 : (dLatc   <   40 ? 4 : 8))) ,
    (diskQ   < 20   ? 1 : (diskQ   <  40 ? 2 : (diskQ   <   80 ? 4 : 8))) ,
    (dOIO    < 20   ? 1 : (dOIO    <  40 ? 2 : (dOIO    <   80 ? 4 : 8))) ,
    (PageO   < 1000 ? 1 : (PageO   <2000 ? 2 : (PageO   < 4000 ? 4 : 8)))
  ])",
```

Note the entire super metric does not have error handling. If a metric does not collect, the entire super metric will return blank.

Pretty cool isn't it? If you agree, send your thanks to Gautam Kumar and Artavazd Amirkhanyan.

VM Uptime

Use Case: calculate the VM uptime within the 5-minute collection cycle.

This particular super metric wasn't fully implemented in the product due to the false positive from the raw vCenter counter that was discovered during validation. So I'm providing as an example of what you can do with super metric.

The up time of a VM is more complex than that of a physical machine. Just because the VM is powered on, does not mean the Guest OS is up and running. The VM could be stuck at BIOS, Windows hits BSOD or Guest OS simply hang. This means we need to check the Guest OS. If we have VMware Tools, we can check for heartbeat. But what if VMware Tools is not running or not even installed? Then we need to check for sign of life. Does the VM generate network packets, issue disk IOPS, consume RAM?

Another challenge is the frequency of reporting. If you report every 5 minutes, what if the VM was rebooted within that 5 minute, and it comes back up before the 5th minute ends? You will miss the fact that it was down within that 5 minutes.

From the above, we can build a logic:

```
If VM Powered Off then Return 0. VM is definitely down.
Else Calculate up time within the 300 seconds period.
```

In the above logic, to calculate the up time, we need first to decide if the Guest OS is indeed up, since the VM is powered on.

We can deduce that Guest OS is up is it's showing any sign of life. We can take Heartbeat from Tools. What if there no Tools or Tools not returning heartbeat? In this case, none of the metrics from Windows/Linux will be available in VCF Operations, unless we install another agent (e.g. Telegraf). We need to have fail back plan. So we check memory usage, network usage and Disk IOPS.

Can you guess why we can't use CPU Usage?

VM does generate CPU even though it's stuck at BIOS. We need a counter that shows 0, and not a very low number. An idle VM is up, not down.

So we need to know if the Guest OS is up or down. We are expecting binary, 1 or 0. Can you see the challenge here?

Yes, none of the metrics above is giving you binary. Disk IOPS for example, can vary from 0.01 to 10000. The "sign of life" is not coming as binary.

We need to convert them into 0 or 1. 0 is the easy part, as they will be 0 if they are down.

I'd take Network Usage as example.

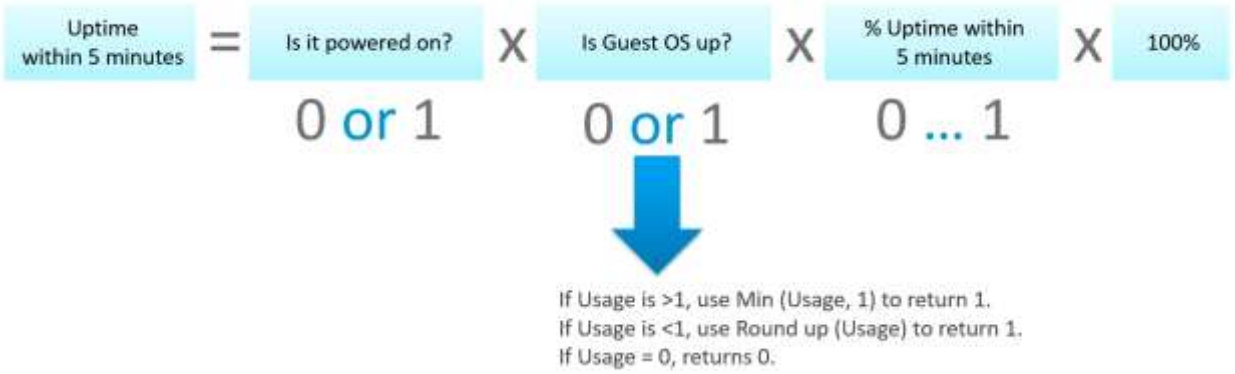
- What if Network Usage is >1? We can use Min (Network Usage, 1) to return 1.
- What if Network Usage is <1? We can use Round up (Network Usage, 1) to return 1.

So we can combine the above formula to get us 0 or 1.

The last part is to account for partial up time, when the VM was rebooted within the 300 seconds sampling period. The good thing is VCF Operations tracks the OS up time every second. So every 5 minutes, the value goes up by 300 seconds. As VM normally runs >5 minutes, you end up with a very large number. Our formula becomes:

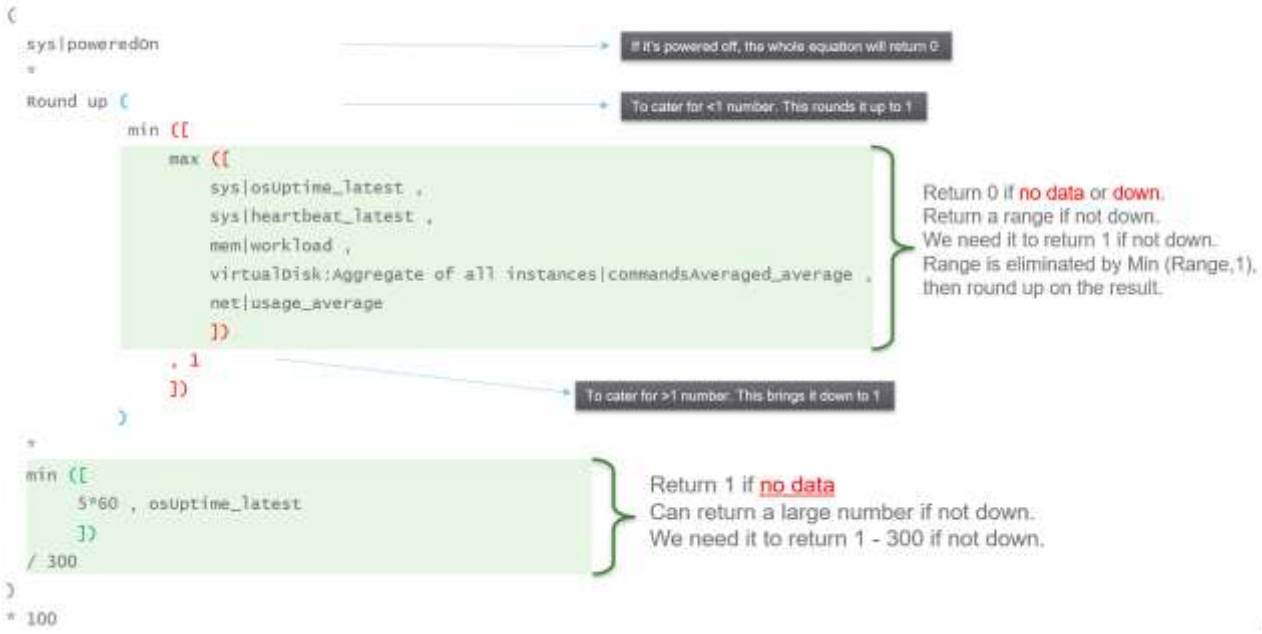
If the up time is >300 seconds then return 300 else return it as it is.

Let's now put the formula together. Here is the logical formula:



Can you write the above formula differently? Yes, you can use If Then Else. I do not use it as it makes the formula harder to read. It's also more resource intensive.

Let's translate the above into a pseudocode.

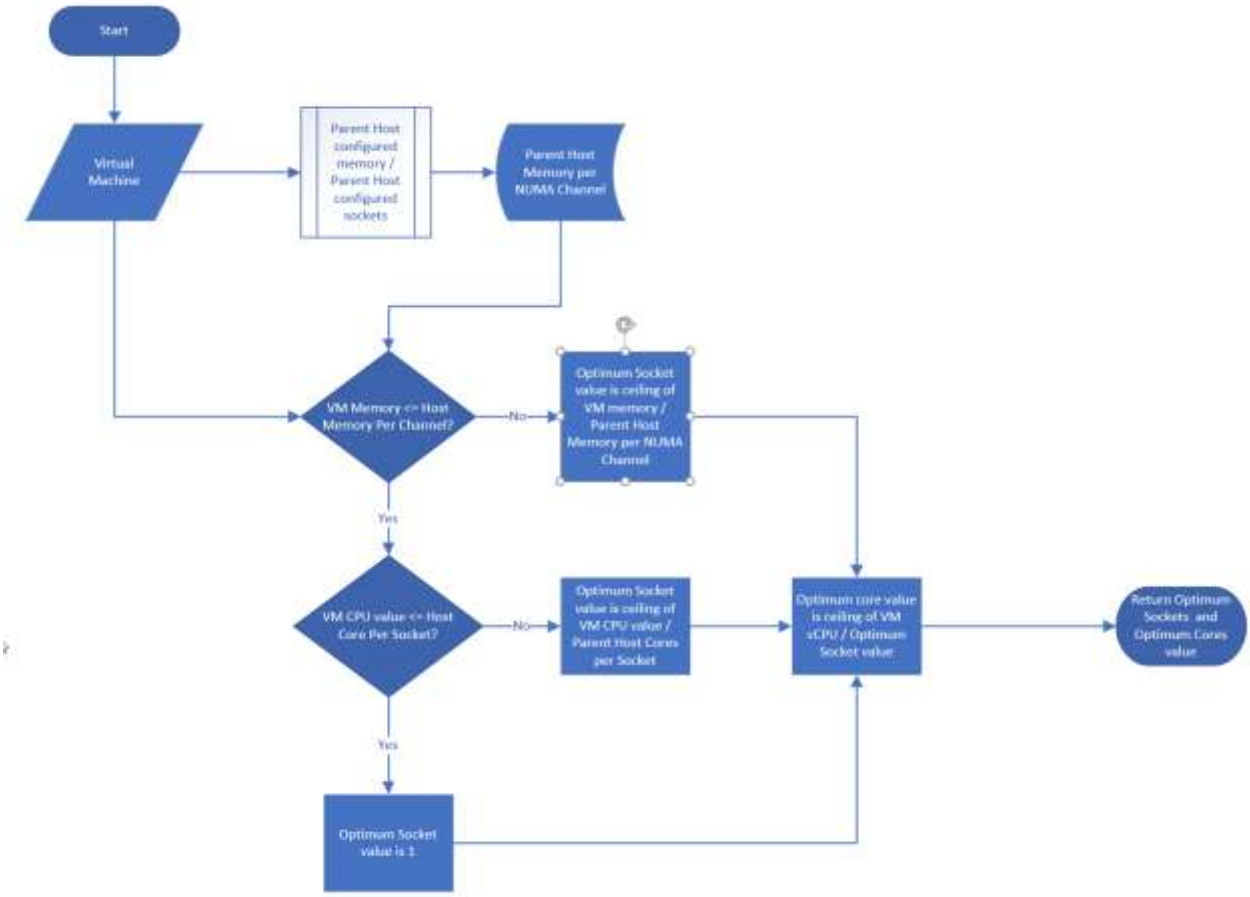


Lastly, here is what it looks like in actual code. I've optimized the last bit to **/3**. No point multiply by 100 then divide by 300.

NUMA Sizing

After VM vCPU is sized, it needs to be validated against the underlying ESXi to consider NUMA. John Diaz explains the use case here, so please read it first if you are unaware of the need for NUMA-compliant sizing.

What I'd like to highlight is John took the time to create a logical design. He visualized it on a flowchart, which I think it's useful to ensure that the logic is bullet proof before you start coding.



John also uses a code editor to help check the syntax. I myself have applied the same technique for long super metric.

```
ceil({This Resource:config|hardware|numCpu})/(({This Resource:config|hardware|memoryKB}/1024/1024) <= (sum({Host System:
Hardware|Memory Size, depth=-1})/1024/1024)/sum({Host System:Hardware|CPU Information|Number of CPU Packages, depth=-1})
?
  (({This Resource:config|hardware|numCpu} as vmCPUs) % 2
  ?
    vmCPUs
    :
    vmCPUs + 1) <= (sum({Host System:Hardware|CPU Information|Number of CPU Cores
, depth=-1}) / sum({Host System:Hardware|CPU Information|Number of CPU Packages, depth=-1}))
  ?
    1
    :
    ceil(vmCPUs/(sum({Host System:Hardware|CPU Information|Number of CPU Cores, depth=-1}) / sum({Host System:Hardware|CPU
    Information|Number of CPU Packages
, depth=-1})))
  : ceil(({This Resource:config|hardware|memoryKB}/1024/1024) / ((sum({Host System:Hardware|Memory Size, depth=-1})/1024/
  1024)/sum({Host System:Hardware|CPU Information|Number of CPU Packages
, depth=-1})))
```

Automation

Part 4 Chapter 2

I bet you still remember that part of Iwan's book: "*Yes, I did not cover **Automation**. For me, that's part of Architecture. You should not automate what you cannot operate.*"

In other words, **what you can operate should be automated**.

It might surprise you but **Aria Operations** is doing automation out of the box in the form of automated control routines; Aria Operations is "automatically" monitoring the environment through predefined or custom **Symptoms**, **Alert Definitions** and **Compliance Packs**. These routines control triggering of **Alarms** and provide information presented through **Dashboards** and **Reports**.

Continuous Performance Optimization cares for best possible performance at minimal cost, driven by **Operational Intent** and **Business Intent** settings with ML supported predictive analytics driving actions to balance workloads and proactively avoid contention. The **Automated Workload Balancing** may help reduce software license costs, optimize performance based on performance or enforce compliance.

Various Compliance Packs help reduce risk and enforce IT and regulatory standards for VMware VCF based private clouds through continuous checks and automated drift remediation.

But there is more.

And this is exactly how you should operate your environment.

Why Automation?

Automation has profoundly impacted the industry since the advent of industrial robotics in the 1950s and 60s. But why is automation important, and what justifies the introduction of a seemingly complex and potentially costly technology?

Indeed, implementing automation requires an initial time investment, which translates to increased CAPEX. However, the benefits are substantial:

- **Reduction in OPEX**: Repetitive tasks no longer need manual intervention, leading to operational cost savings.
- **Resource Optimization**: Automation frees up time, allowing focus on more critical business activities.
- **Consistency and Accuracy**: Automated processes follow a predefined procedure, minimizing human errors (excluding procedural errors).
- **Ease of Auditing**: Automation simplifies or even enables the auditing process.
- **Enhanced Productivity, Reliability, and Performance**: Automation drives improvements across these key areas.

Understanding these benefits highlights why automation is a valuable investment for any business.

Basic Principles of Automation

Before diving into the essentials of this chapter, let's briefly examine the typical components of automated systems. The following image illustrates the four fundamental elements of an automated system: a **controller**, **sensors**, **actuators**, and the **plant** we want to automate, all forming a closed loop.

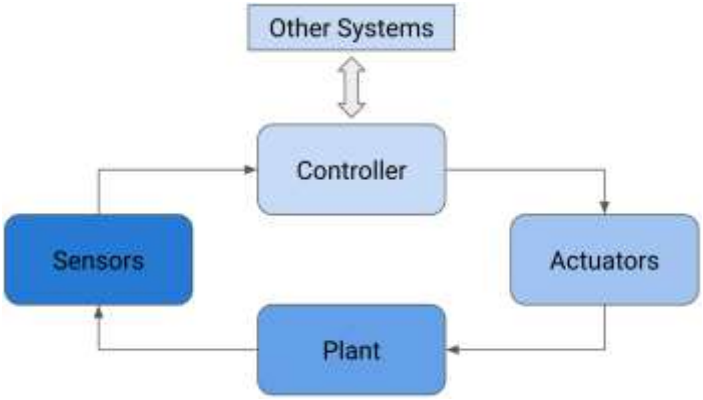


Figure 1: Components of an automated system in a closed-loop feedback control system.

This is example of a closed loop is also called **closed-loop feedback control**. Another, less complex kind of automated system is shown in the next picture. It is a so-called **open-loop control**.

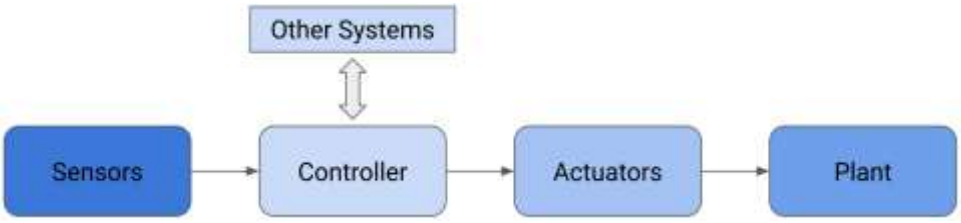


Figure 2: Open-loop control system.

How does this apply to an environment managed by Aria Operations? Let's begin with a simpler example, as the core components remain consistent in both cases. As shown in the next figure, certain parts can be easily mapped to components found in your Software-Defined Data Center (SDDC) run on VMware VCF.

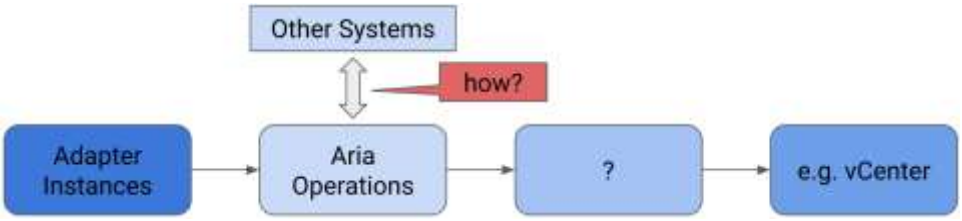


Figure 3: Aria Operations in an open-loop control system.

In this context, the **sensors** are our **adapter instances** that collect metrics and properties, the **controller** is **Aria Operations** itself, and the **plant** we aim to control or automate is, for example, our VMware **vCenter** instance or instances managing ESXi hosts, VMs, and Datastores.

But what are the **actuators**, and how can the controller (the Aria Operations instance) interact with other systems?

I will address both questions in the following subsections. So keep reading—you've made it through the theoretical part.

Aria Operations and Automation

Let's delve into the second part of the question: how can Aria Operations interface with other systems?

When referring to "other systems," I mean systems that contribute to automation but are not necessarily the target plant for control. It's not surprising that Aria Operations can interact with other systems using interfaces provided by those systems. In the realm of automation, we commonly utilize Application Programming Interfaces (APIs). Interacting with other systems involves both outbound and inbound communication from the perspective of Aria Operations.

Outgoing Communication

In this section I will describe the available options to communicate with 3rd party APIs and send information from Aria Operations to another, so called north-bound, systems.

When speaking about north-bound systems we usually refer to services like mail servers or SNMP trap receivers. Since the advent of 8.x versions of Aria Operations, the product supports more REST services like ServiceNow and Slack natively.

In the next subsection, I will describe the configuration of the commonly used north-bound outgoing connectivity.

Automated Action Plugin

In Aria Operations, there is a selection of **Outbound Plugins** designed for creating **Outbound Instances**. The following image displays the available plugins. To illustrate this concept, the Standard Email Plugin is utilized to configure Aria Operations for sending emails via SMTP. An Outbound Instance based on this plugin specifies a particular mail server for sending emails, along with various delivery parameters.

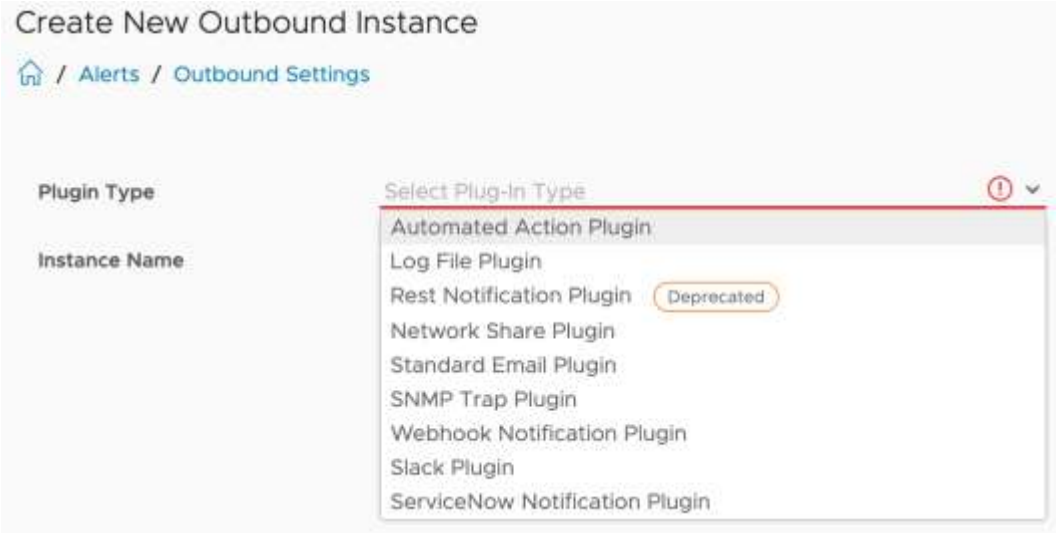


Figure 4: Aria Operations Outbound Plugins.

A configured Outbound Instance provides a one-way communication towards another system.

The first plugin type in this list is the **Automated Actions Plugin**. After initial installation and configuration of Aria Operations, an Outbound Instance based on this plugin is automatically available for consumption.

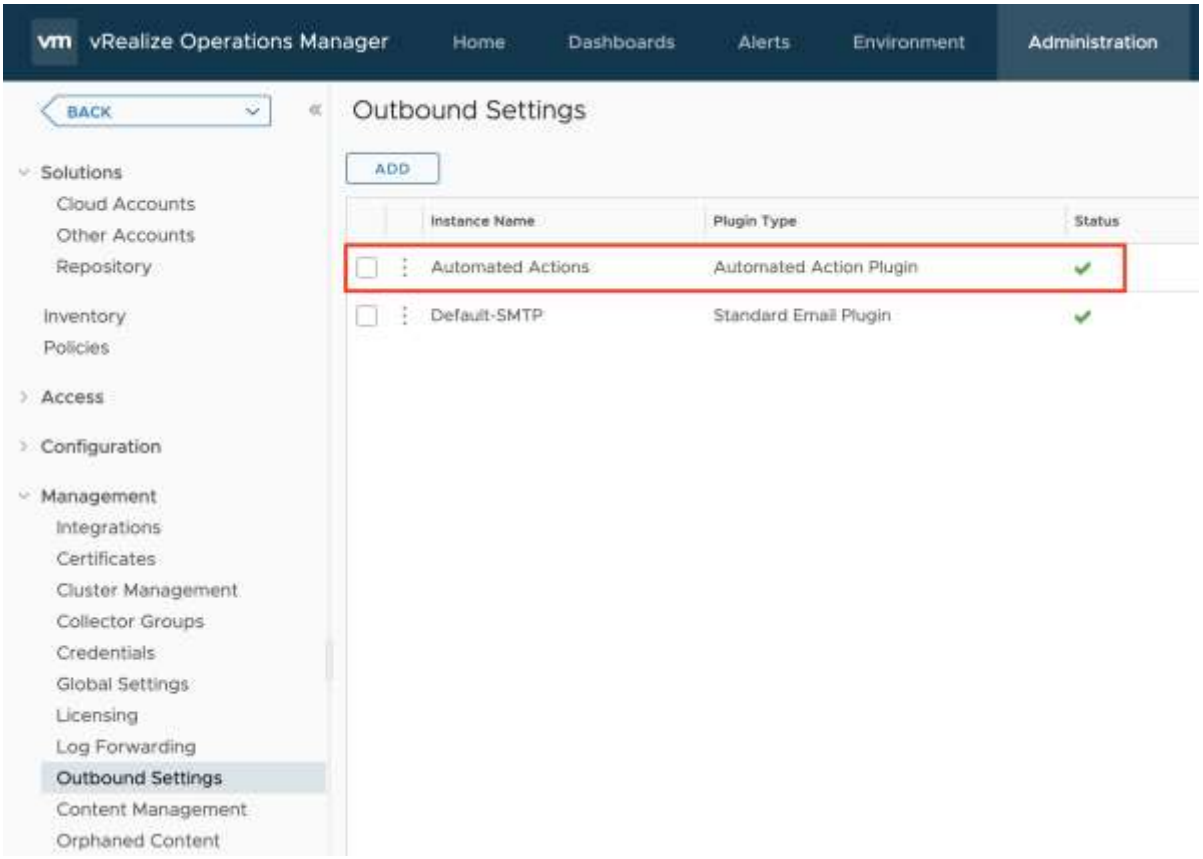


Figure 5: Automated Actions Outbound Instance.

This particular Outbound Instance enables Aria Operations users to perform actions on managed objects living in the connected vCenter instances directly from Aria Operations, as shown in the next screenshot. The actions available to us after the installation are part of the respective Management Packs.

The available actions in the Actions drop-down menu depend on the currently selected object and its type. The list varies, for example, between virtual machines and datastore objects.

In the following image, the actions marked with a small yellow icon are Aria Automation Orchestrator workflows that have been configured out-of-the-box.

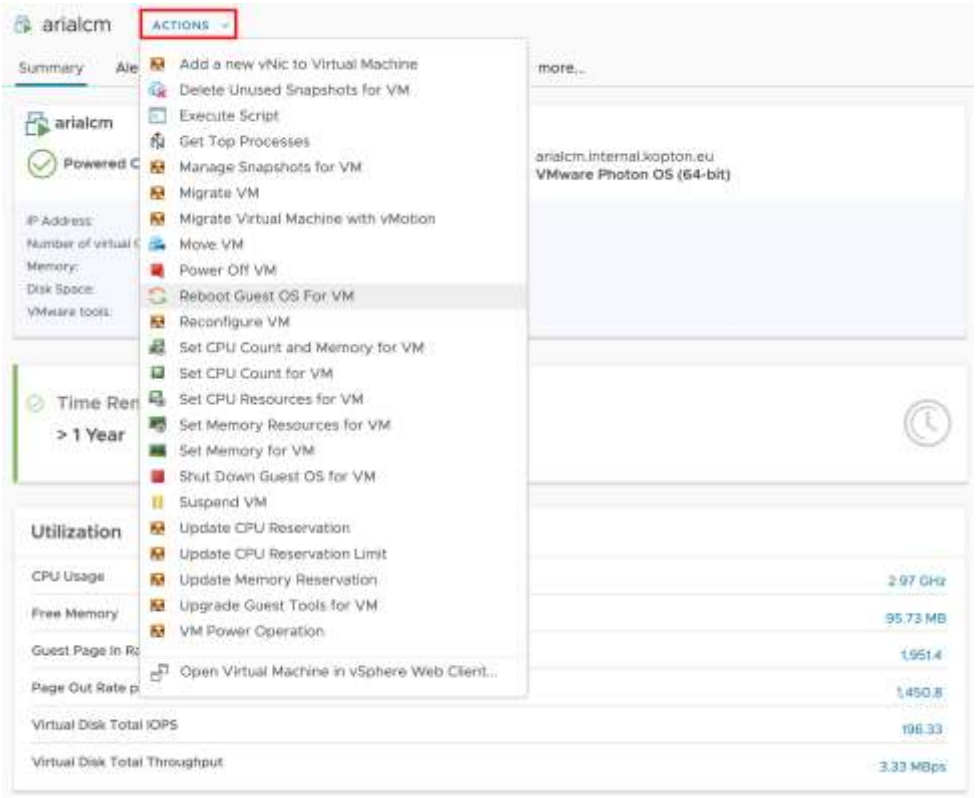


Figure 6: Available Automated Actions.

A prerequisite to enable such automated actions is enabling the **Operational Actions** setting in the configuration of the corresponding vCenter **Cloud Account** adapter instance and sufficient permissions granted to the account used for the vCenter connection.

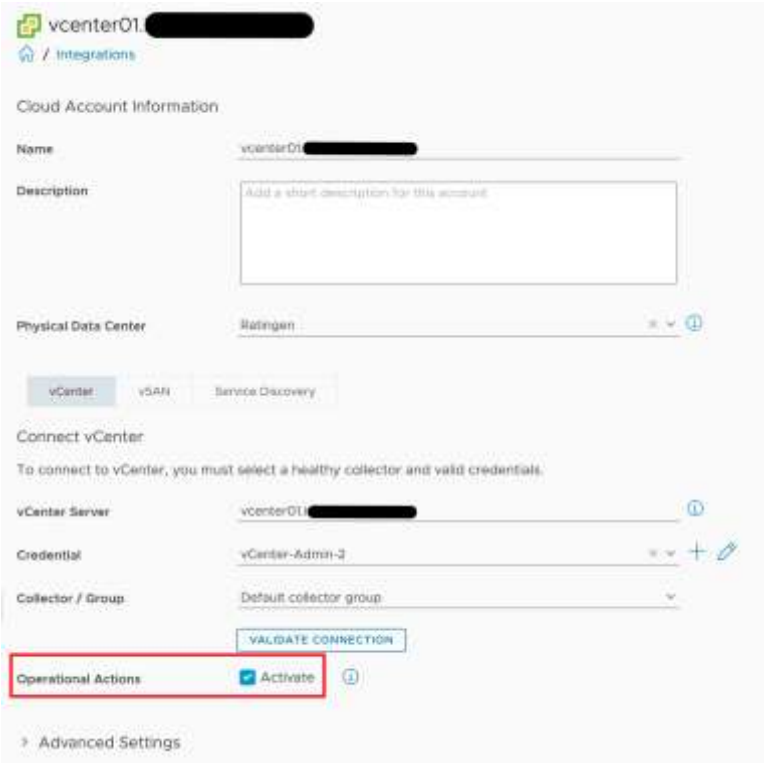


Figure 7: Operational Actions option of vCenter Adapter Instance.

Automated Actions, as the name implies, can be used to automate execution of tasks as part of a **Recommendation** within an Aria Operations **Alert Definition**.

Out-of-the-box Aria Operations provides several Recommendations containing vCenter actions, like shown in the following picture. The automated execution of such actions is enabled through Aria Operations Policies.

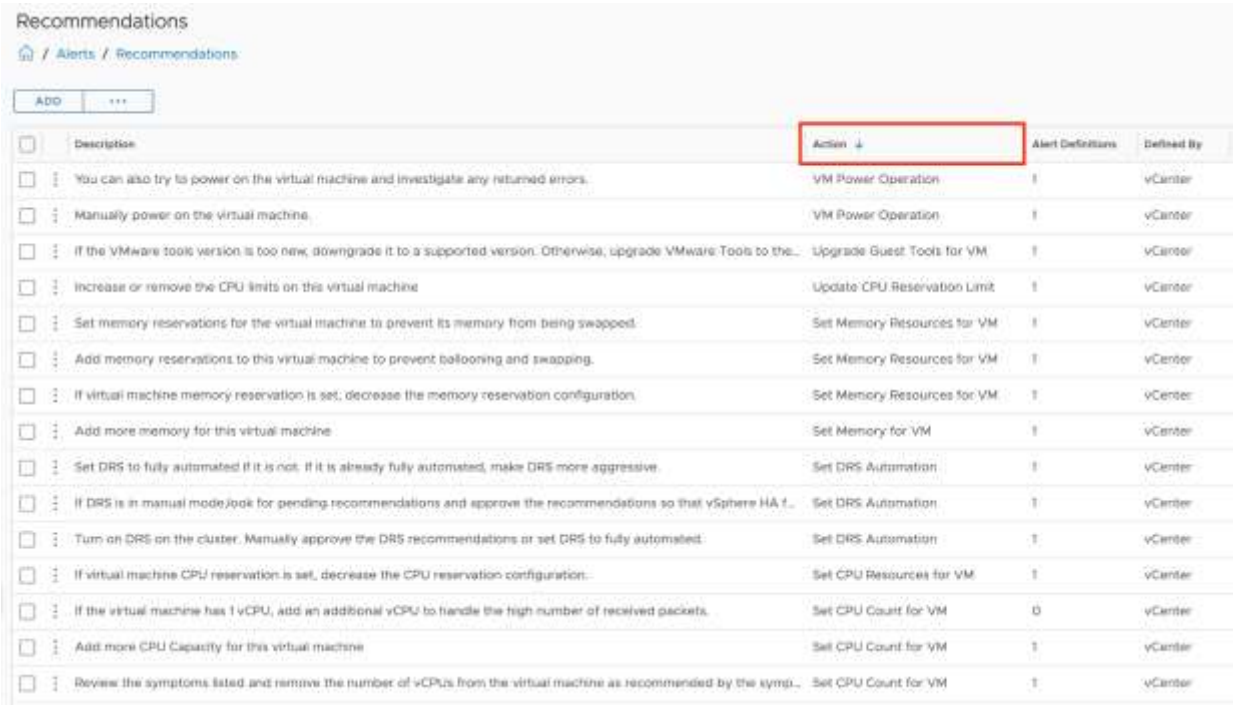


Figure 8: Actions and part of Recommendations.

The initial step towards achieving a **Self-Healing Software-Defined Data Center** involves automating these actions. However, in an SDDC, simply enabling automation is not sufficient. For instance, compliance with change management processes, involvement of VM owners, or obtaining approvals are necessary considerations. How to exactly address these requirements is not covered in this chapter, but you will explore the available options.

Now, before I provide a brief guide on enabling action automation with a simple example, let me briefly explain the **Alert Definition** concept in simplified terms.

The following diagram illustrates the relationships between the relevant objects. An Alert Definition can have one or multiple Recommendations. Each Recommendation may include an Action, which becomes accessible within the Recommendation and can be executed manually from the details view of an alert.

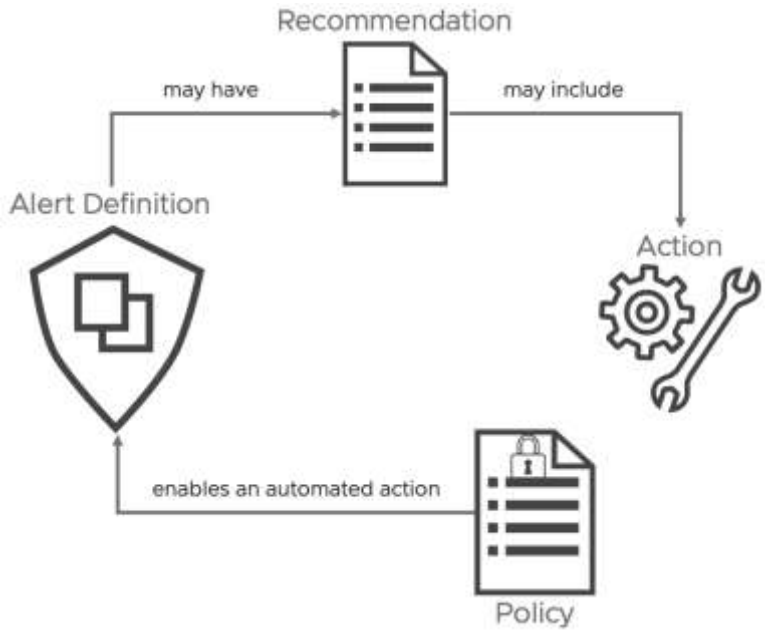


Figure 9: Simplified Alert Definition concept.

Now, there's just one more step needed to implement automation: enabling the automated execution of the action within the corresponding Aria Operations Policy that governs the behavior of each object. The following image illustrates this within my default policy, specifically in the Alert Definition section.

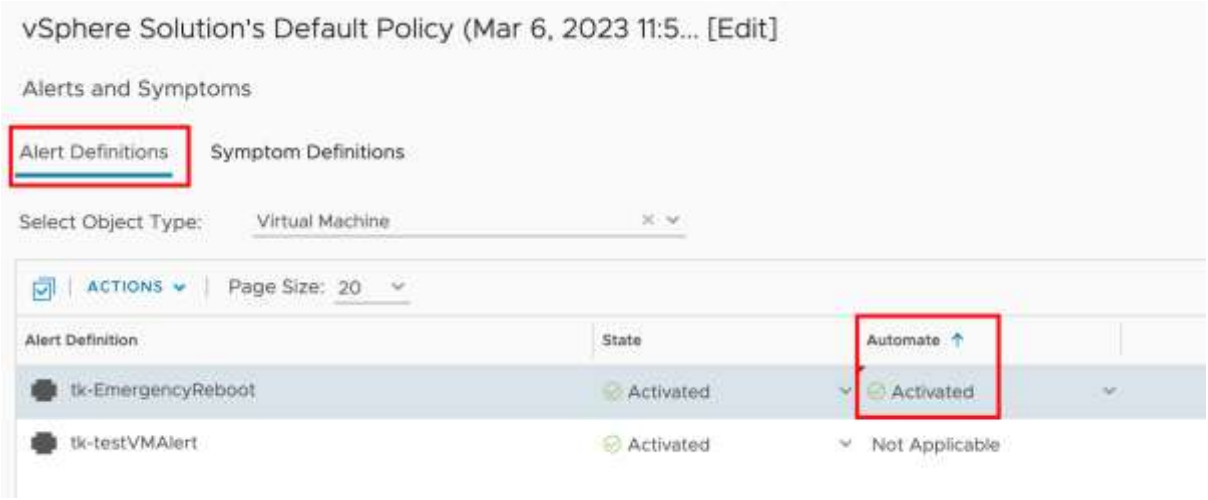


Figure 10: Enabling automated Actions in a Policy.

The next time this specific alert is triggered, Aria Operations will attempt to automatically execute the configured action. I use "attempt" because certain actions may require specific inputs and cannot run without interaction from the administrator.

Over time, the process of creating an Alert Definition in Aria Operations has become significantly clearer. The following image shows the individual steps, including the (optional) configuration of Alert Recommendations which may include Recommendations Actions.

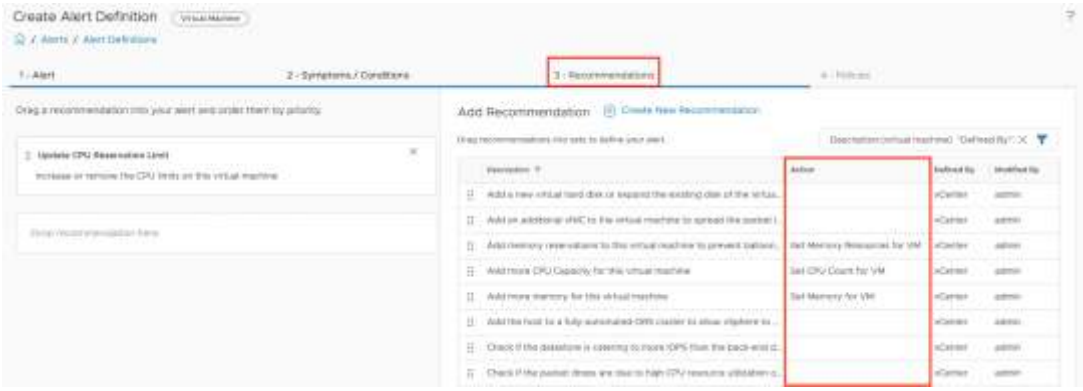


Figure 11: Recommendations option in Alert Definition.

The Automated Action Plugin is a convenient and user-friendly option for automating tasks triggered by Aria Operations alerts, but it has its limitations. The available actions are pre-configured and cannot be customized to address more complex scenarios or additional object types.

I will describe later in detail how to "overcome" the lack of flexibility of the pre-configured actions.

SNMP Trap Plugin

Another plugin that can automate tasks triggered by Aria Operations is the **SNMP Trap Plugin**. SNMP **traps** are typically set up to trigger alerts or open tickets through systems equipped with an SNMP **receiver**. By leveraging **VMware Aria Automation Orchestrator** as an SNMP Trap receiver, you can incorporate SNMP traps into your automation workflows. In the following image, you can view the configuration of an SNMP Outbound Instance, which is used to send SNMP traps to a VMware Aria Automation Orchestrator endpoint. In this example, the endpoint is the integrated Aria Automation Orchestrator instance.

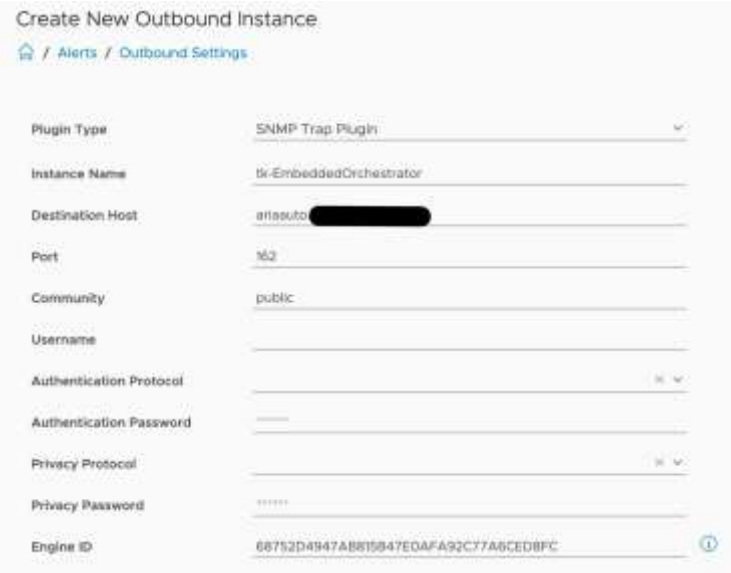


Figure 12: SNMP-Plugin Outbound Instance.

This link describes how to configure a **SNMP Trap Receiver Policy** in VMware Aria Automation Orchestrator which will be able to start scripts or workflows anytime a SNMP trap has been received. You can also learn how to set up all needed components to start an automated workflow using the SNMP Trap Plugin reading this blog post

Webhook Notification Plugin

The new **Webhook Notification Plugin** offers a versatile method to integrate with any REST API endpoint.

As we have already learned, we must first create a suitable Outbound Instance to establish a connection with a 3rd party REST API in order to invoke the methods provided there.

We'll look at an example that one of my esteemed colleagues worked on.

John Dias developed a sample Webhook for integrating with Microsoft Teams. To use MS Teams as a target for Aria Operations Notifications, you must first create an incoming Webhook connector for your Teams channel. The configuration of the Webhook in MS Teams is not covered in this chapter, but it is thoroughly explained in John's post.

To utilize the MS Teams connector in Aria Operations, simply create an Outbound Notification Plugin Instance in Aria Operations with the complete URL of the MS Teams incoming webhook as a parameter, as demonstrated in the following image:

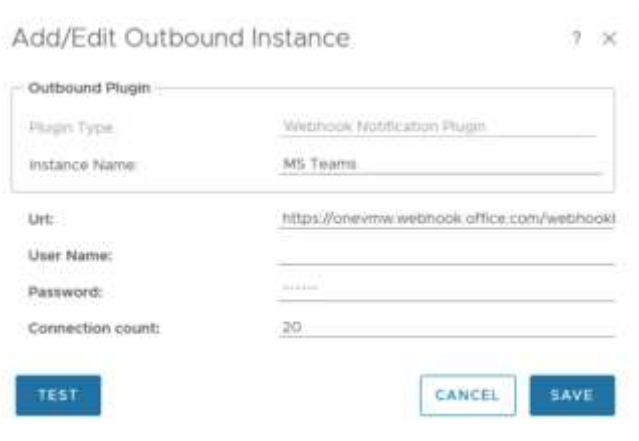


Figure 13: Webhook Notification Outbound Instance for MS Teams.

The supplied payload example simplifies the creation of a Notification. The **Payload Template** defines the information that Aria Operations will transmit to the target when the specified notification rules are met. The following figure displays the template that John created for the MS Teams integration.

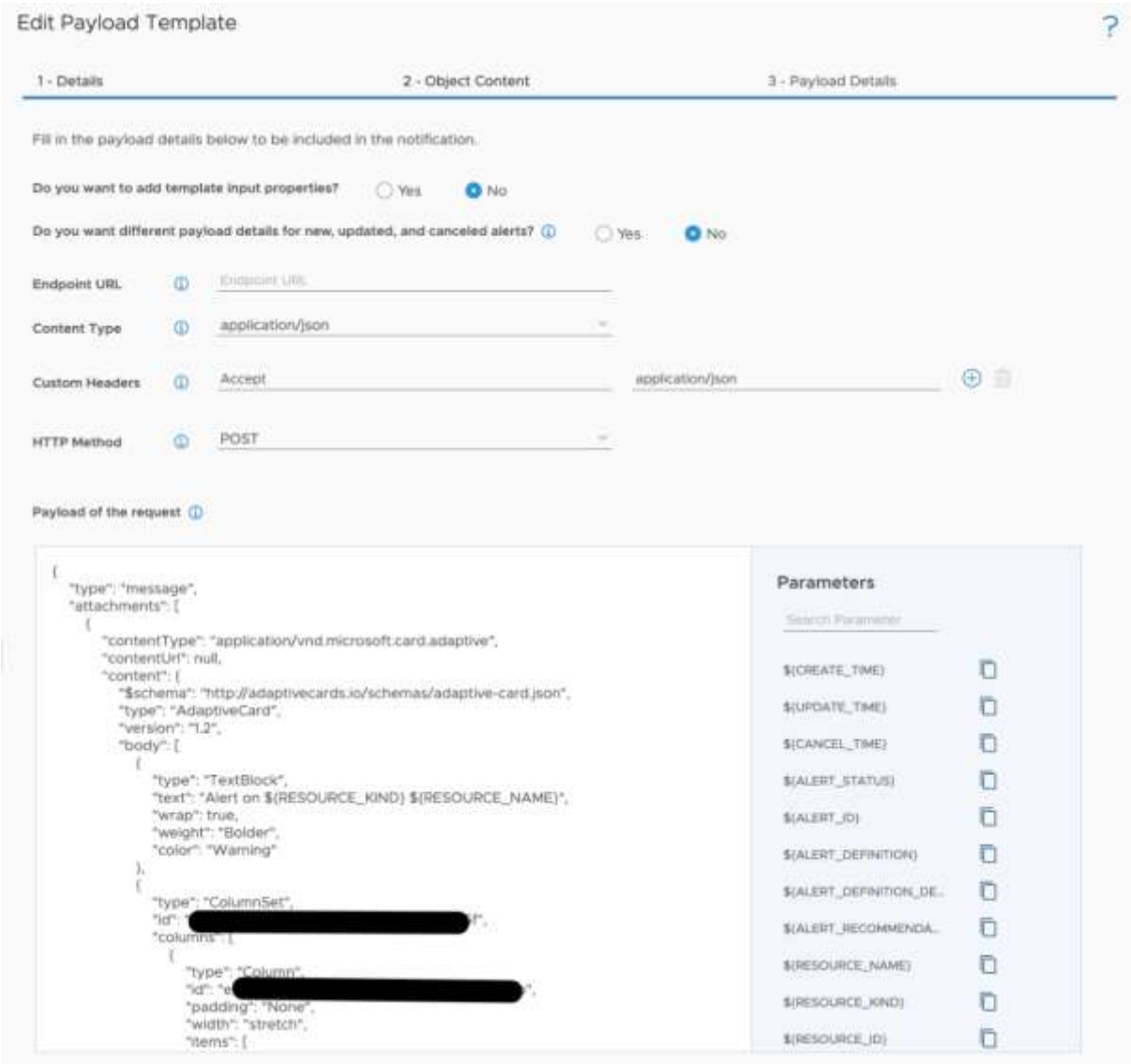


Figure 14: Webhook Payload Template for MS Teams.

Utilizing Notifications, the Webhook Notification Outbound Instance, and the new **Payload Templates**, you can finely tune the desired outbound payload down to specific metric levels. The following image demonstrates an example of the payload configuration applicable to a webhook outbound connection.

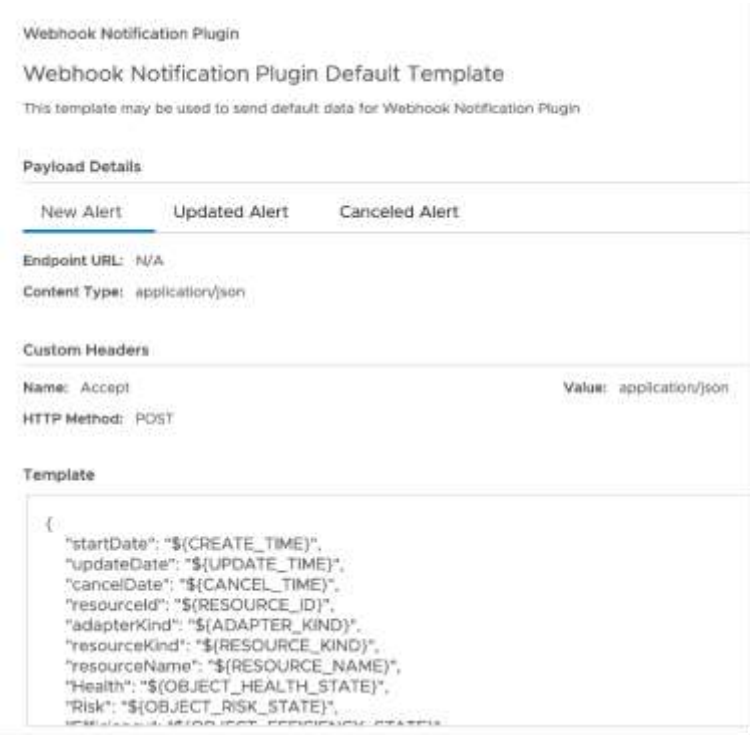


Figure 15: Webhook Payload Template general example.

The enhanced Webhook capabilities enable Aria Operations to integrate with systems that were previously not within our scope, such as for example message queues.

By configuring a Webhook Notification Plugin Instance to connect to a **RabbitMQ** endpoint and using an appropriate payload template, Aria Operations now supports creating notifications that send data to RabbitMQ queues based on specific events. For instance, this example demonstrates RabbitMQ queues for opened and cancelled alerts.

The prerequisite for this use case is, of course, a functional RabbitMQ instance with the following queues that will receive our messages.

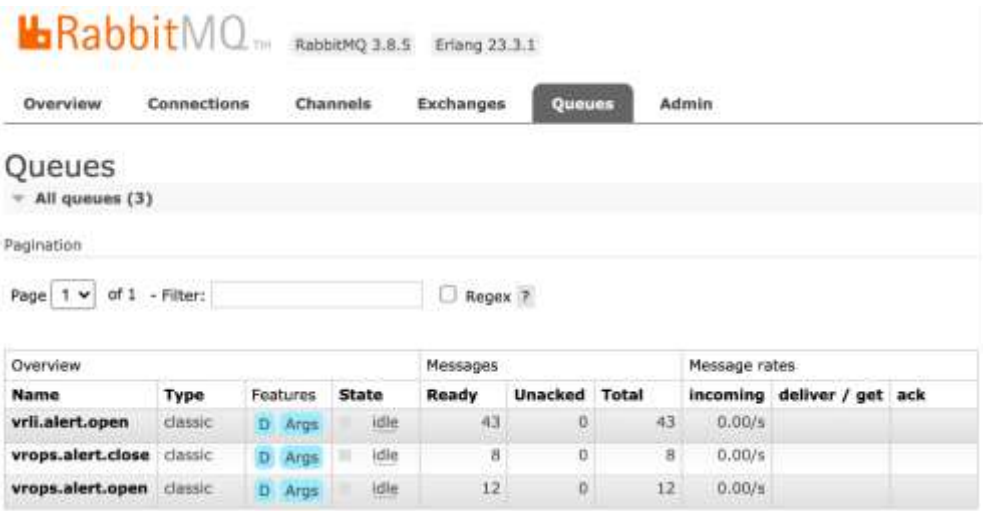


Figure 16: RabbitMQ queues.

The entire process is described in more detail here.

Basically, on the Aria Operations side, we need an Outbound Instance based on the Webhook Notification Plugin and suitable Payload Templates that will be used in Aria Operations Notifications.

In this context, although technically not necessary we could have different notifications for opening and closing an alert, both using the same Outbound Instance but employing different Payload Templates.

The next picture shows the Outbound Instance I have created to connect to my RabbitMQ server.

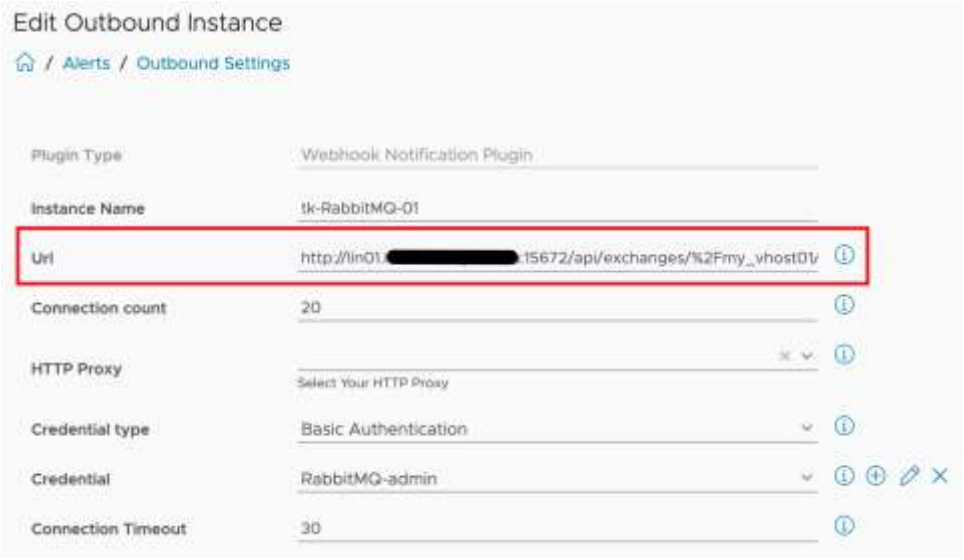


Figure 17: RabbitMQ Outbound Instance configuration.

Please notify that you have to specify the correct host and hosts configured in your RabbitMQ system.

The next image exemplifies the payload template used to place a message into the open queue using an Aria Operations Notification.

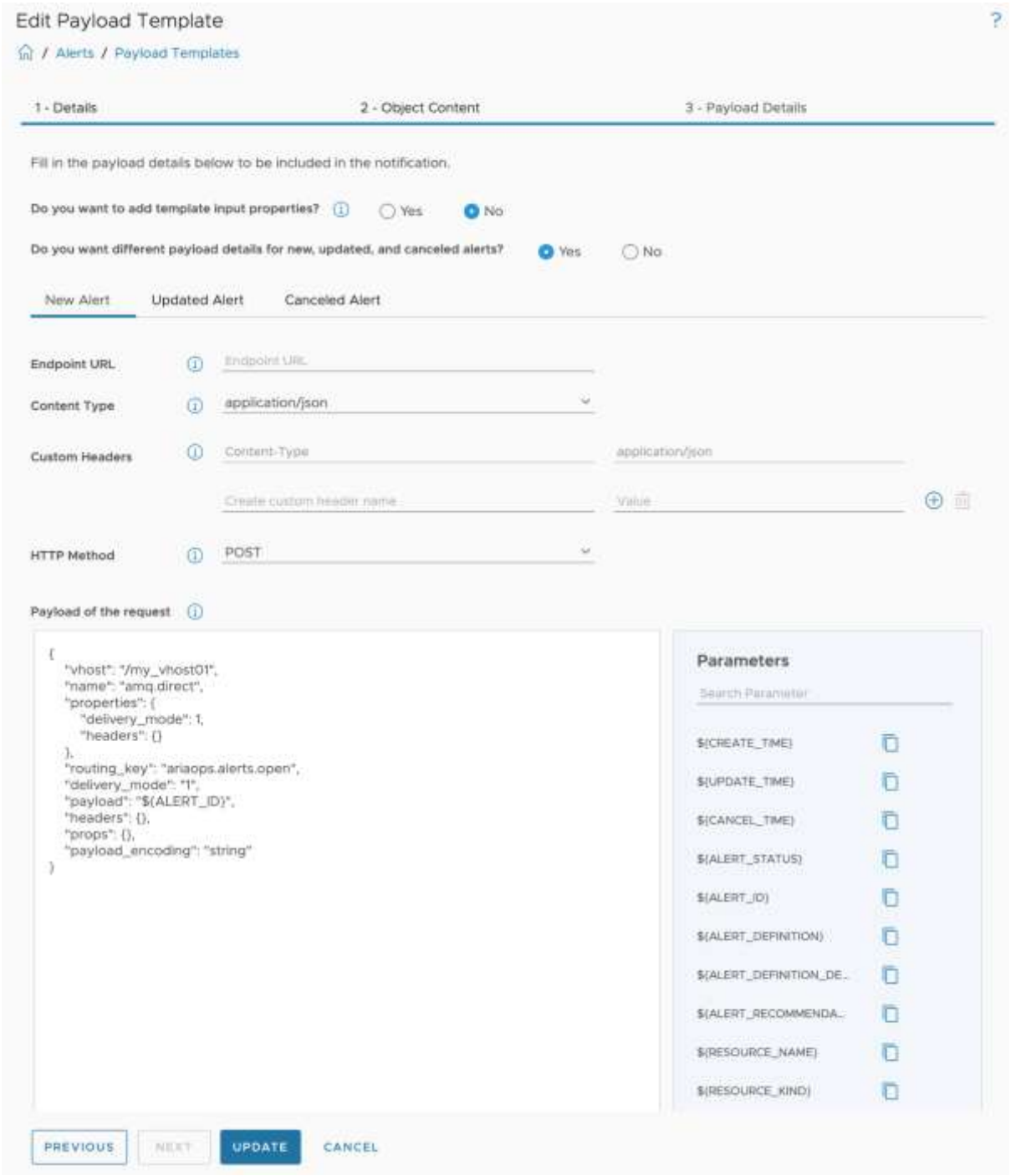


Figure 18: Payload Template for RabbitMQ Outbound Instance.

The payload template offers many configuration options, with the most important being the actual JSON body, especially when discussing REST POST calls. As shown in the last image, in addition to the body, we can configure headers, variables, and other elements typical for REST endpoints.

Notifications

I have mentioned the **Aria Operations Notifications** construct very often without describing what it actually is and how it is configured. I will address that now.

If you look again at the images of an open or closed-loop control system shown at the beginning, you will probably wonder what the role of the Actuators in Aria Operations is supposed to be. That is precisely the task of the Aria Operations **Notifications**.

The **Notification** element provides the required functionality to connect the **sensors** with the **Actuators**. The configuration of Notifications includes not only the rules or criteria for when a Notification should be triggered but also the Outbound Instance to be used and the appropriate Payload for the selected type of Outbound Instance.

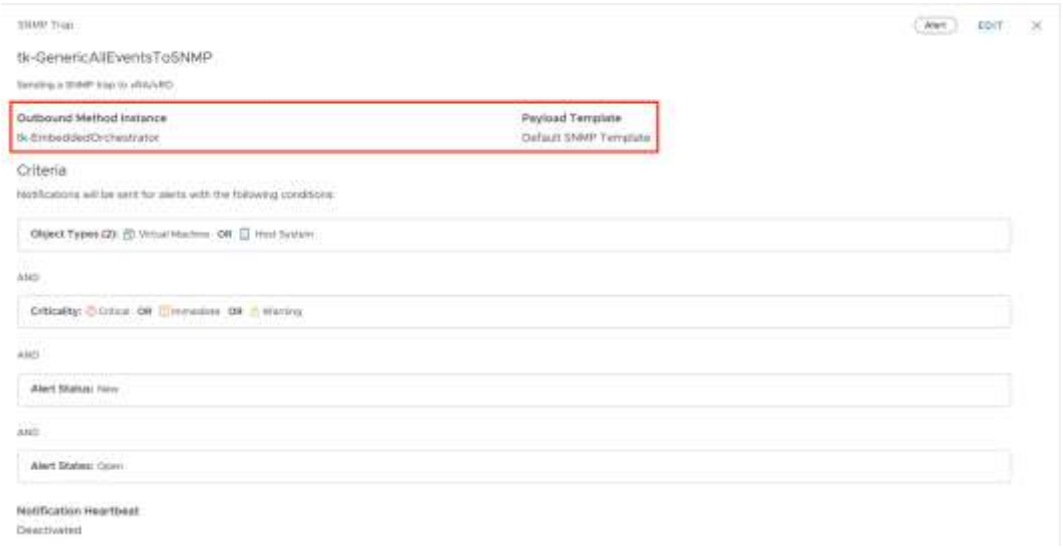


Figure 19: Notification using a SNMP Trap Outbound Instance.

With this information we can extend the model introduced in figure 9 to also include Notifications and Outbound Instances as possible building blocks of automated operations.

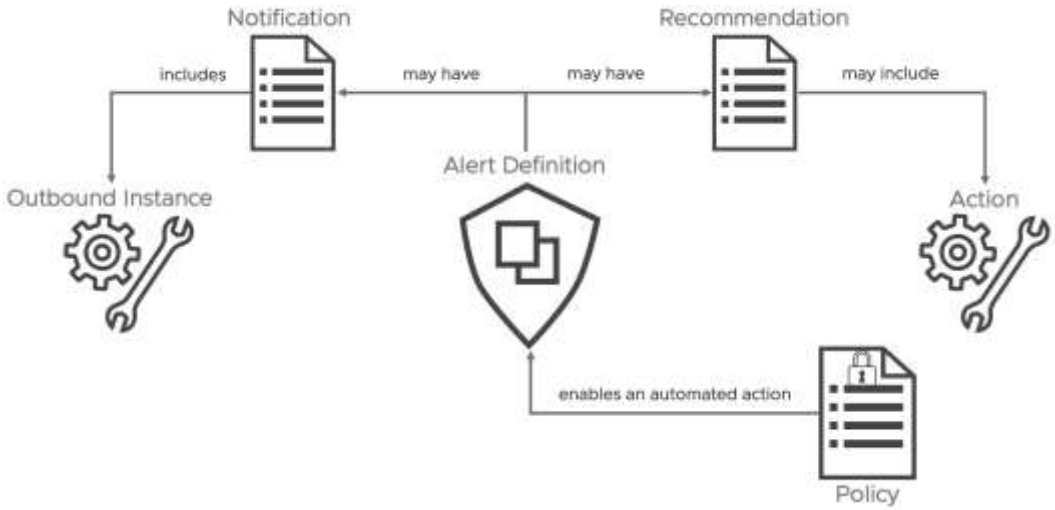


Figure 20: Extended Alert Definition concept.

In the context of automation using Aria Operations, both Automated Actions and Notifications using Outbound Instances exemplify what I would refer to as "fire-and-forget" or, as discussed earlier in this chapter, the **open-loop** approach to executing automation tasks. If your use cases demand more, such as retrieving additional data from Aria Operations to make more informed decisions, continue reading to discover how to effectively tackle these challenges.

Aria Operations Management Pack for Aria Automation Orchestrator

At the end of the "Automated Actions Plugin" chapter, I promised to discuss how the limited flexibility of Automated Actions can be overcome using other methods. In this chapter, I will describe how this can be achieved through the **Aria Operations Management Pack for Aria Automation Orchestrator**. It opens up completely new opportunities for automating your operational tasks and provides real self-healing capabilities. In particular, this Management Pack provides following features:

- Insights into Aria Automation Orchestrator including various metrics, properties and predefined dashboards and alert definitions.
- **Aria Automation Orchestrator workflows available as Actions and within Recommendations.**

With this solution it is easy to implement sophisticated **self-healing** and **self-driving** SDDC, managed by Aria Operations.

The ingredients are:

- **VMware Aria Automation Orchestrator** instance. You can use a stand-alone Aria Automation Orchestrator instance or the instance deployed as part of the Aria Automation installation.
- **Aria Operations**.
- In Aria Operations the **Aria Operations Management Pack for Aria Automation Orchestrator** installed and configured.

To explain the individual configuration steps, I will use a use case that could occur in real operation.

“If a VM (the OS) crashes, this VM should be hard-reset.”

As the determination of whether a VM has actually crashed or the specific process for identifying this is not the focus of this chapter, we will assume that we have appropriate symptom and alert definitions established.

The general approach from use case to automated remediation or any other task is broadly applicable in most scenarios:

- Develop an VMware Aria Automation Orchestrator **workflow** tailored to your specific use case.
- Create or modify an Aria Automation Orchestrator **package** to include your workflow or workflows.
- Incorporate the required workflows into Aria Operations by **discovering** or re-discovering the Aria Automation Orchestrator package created or modified in the previous step.
- **Configure** the workflow or workflows within Aria Operations, this is an important step in the entire process.
- Create or refine Aria Operations **Recommendations**.
- Integrate Aria Operations Recommendations into an Aria Operations **Alert Definition**.
- (Optional) Undertake manual remediation steps if necessary.
- (Optional) Activate automated remediation processes for enhanced efficiency.

In this post, I won't delve into the specifics of the workflow content or coding in Aria Automation Orchestrator. Instead, I'll concentrate on how to seamlessly integrate any workflow into Aria Operations, enabling it to execute automatically as part of the alert remediation process.

For our use case, which focuses on a Virtual Machine, the Aria Automation Orchestrator workflow requires at least one input parameter, in my example it is the `vm` input parameter, to pass the vCenter VM object reference from Aria Operations to Aria Automation Orchestrator. In the image below, you'll also see a second input parameter, `vrops_alert_id`. When this parameter is available, Aria Operations will pass the internal Aria Operations Alert ID to Aria Automation Orchestrator. This ID can be used for callbacks to retrieve more information from Aria Operations. The next subsection of this chapter will explain how to achieve this.

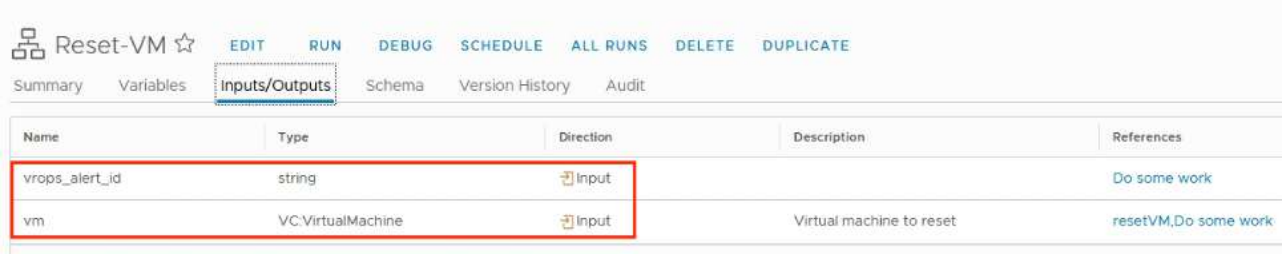


Figure 21: Aria Automation Orchestrator workflow and its input parameters.

The workflow inputs are:

- `vm` as `VC:<Datatype>`, it will be populated with the reference to the VM object which triggered the alert.
- `vrops_alert_id` as `String`, it will be populated with the actual Aria Operations Alert ID for further callbacks.

To use workflows in Aria Operations, they need to be included in a new or existing Aria Automation Orchestrator package. Remember, all workflows within the package that we import in the next step will be visible in Aria Operations. I recommend creating a dedicated package specifically for workflows used in Aria Operations. The next image illustrates a package containing the `Reset-VM` workflow.

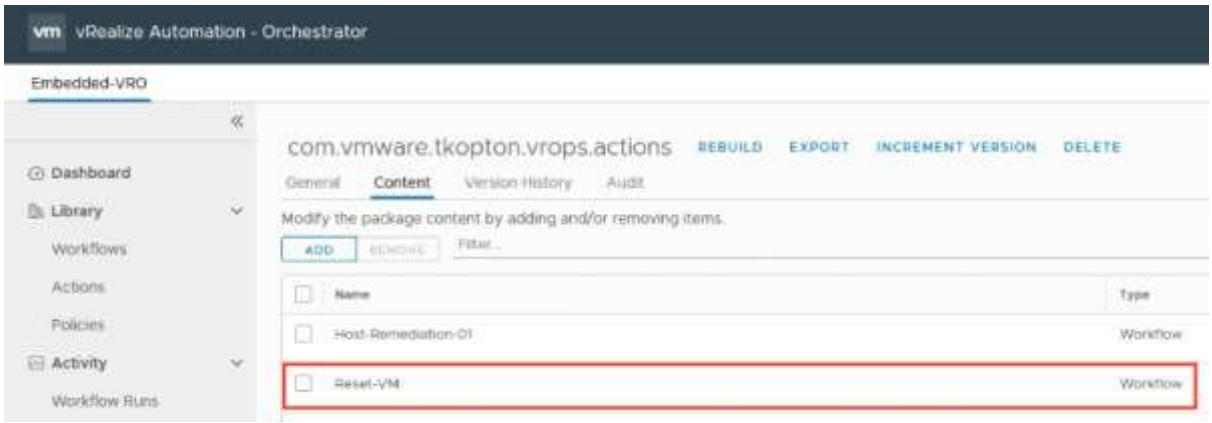


Figure 22: Aria Automation Orchestrator package content.

To make Aria Operations aware of the available workflows, we need to discover the package created or modified in the previous step. This process is straightforward. Navigate to "Environment," then "VMware Aria Automation Orchestrator," and finally "Aria Automation Orchestrator Workflows," as illustrated in the next image.

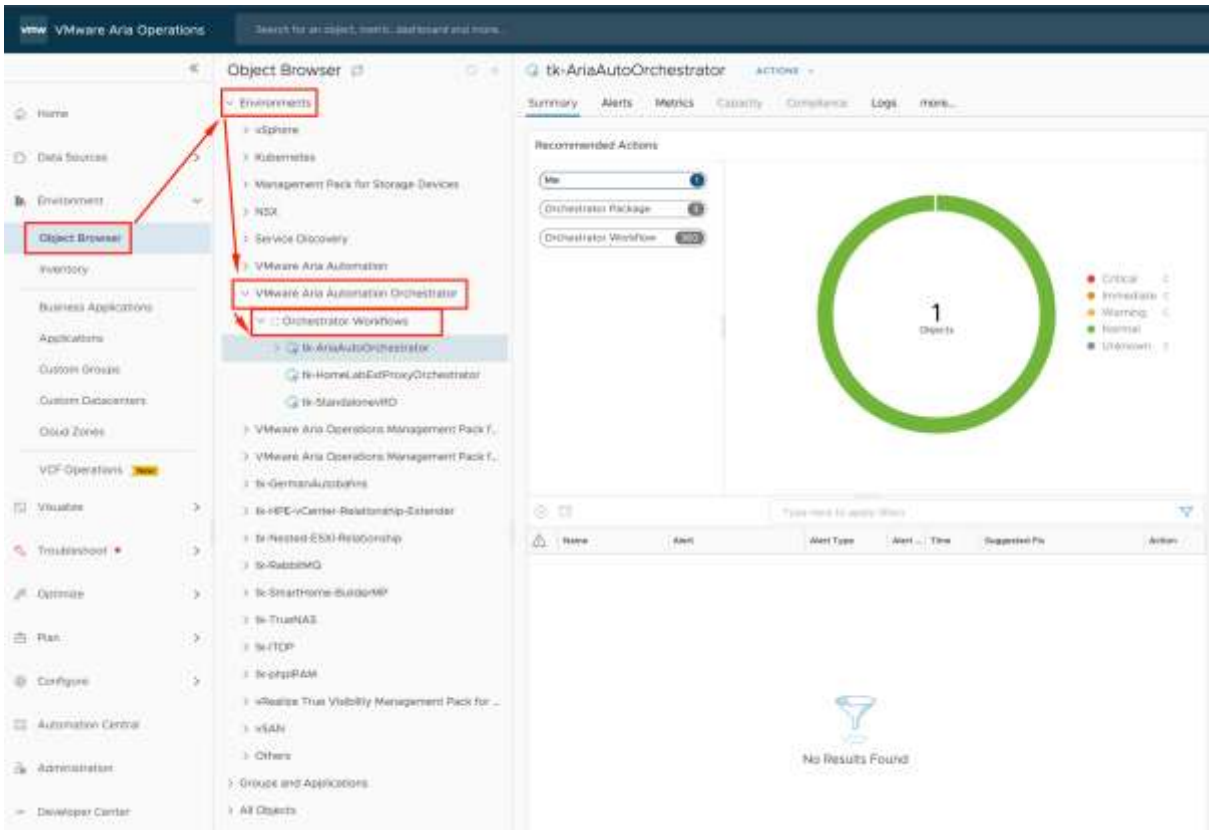


Figure 23: Aria Automation Orchestrator package discovery process.

The next step involves selecting your Aria Automation Orchestrator adapter instance. In my example, this is the integrated Aria Automation Orchestrator, followed by running the "Configure Package Discovery" action from the Actions menu. The subsequent figure illustrates this process.

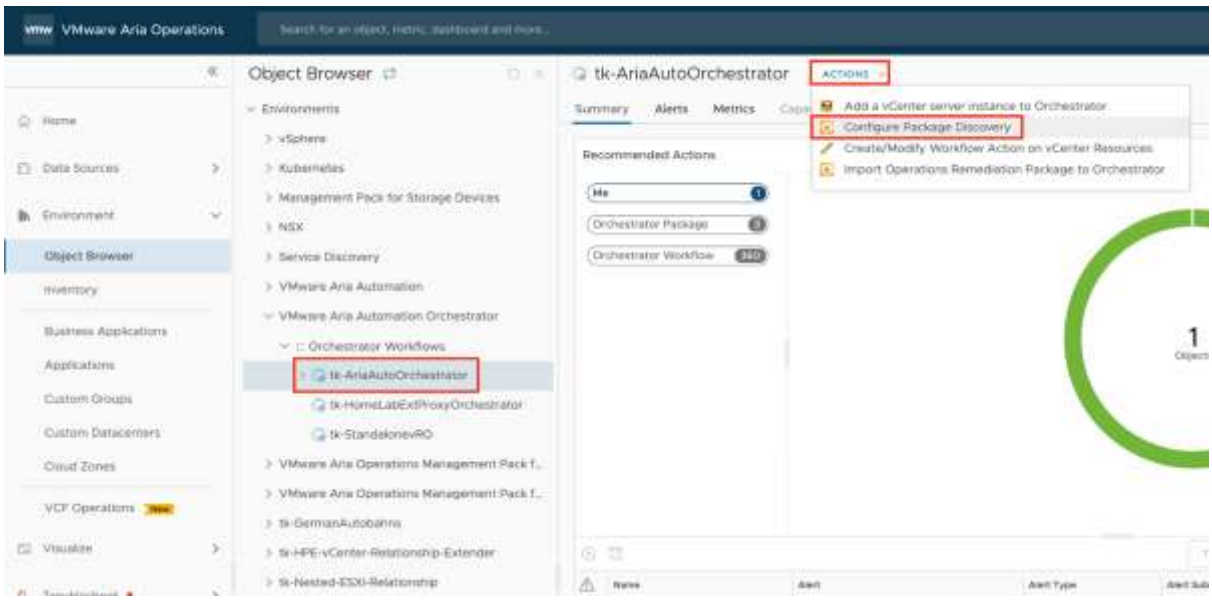


Figure 24: Configure package discovery in Aria Operations.

To initiate the discovery, you'll need to include the relevant package or packages in the list of all packages to inspect. Of course you can remove any package you do not want to be discovered by Aria Operations. Before you remove any package, make sure that none of the included workflows is being used in any of your recommendations. In my example the package I want to scan for workflows is represented by `com.vmware.tkopton.vrops.actions`. The process begins with the "Begin Action," illustrated in the following figure.

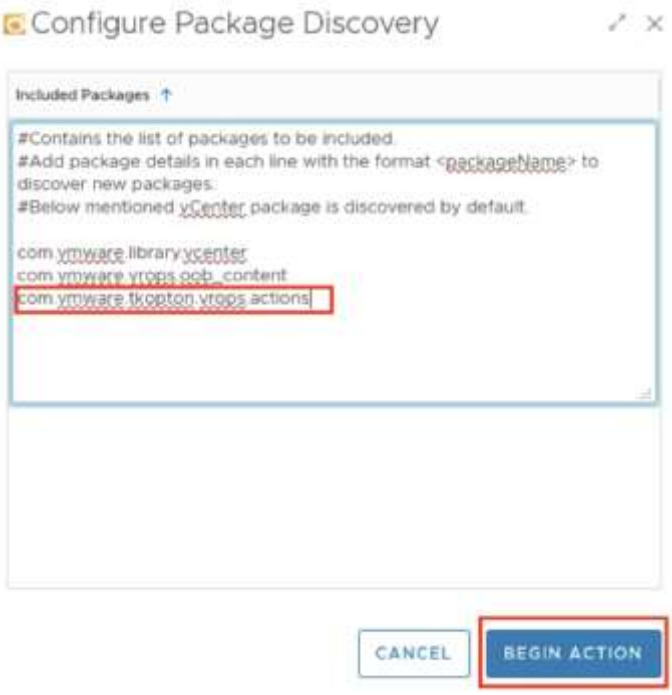


Figure 25: Running the package discovery.

Next picture shows the package and the included workflows after successful completion of the discovery process.

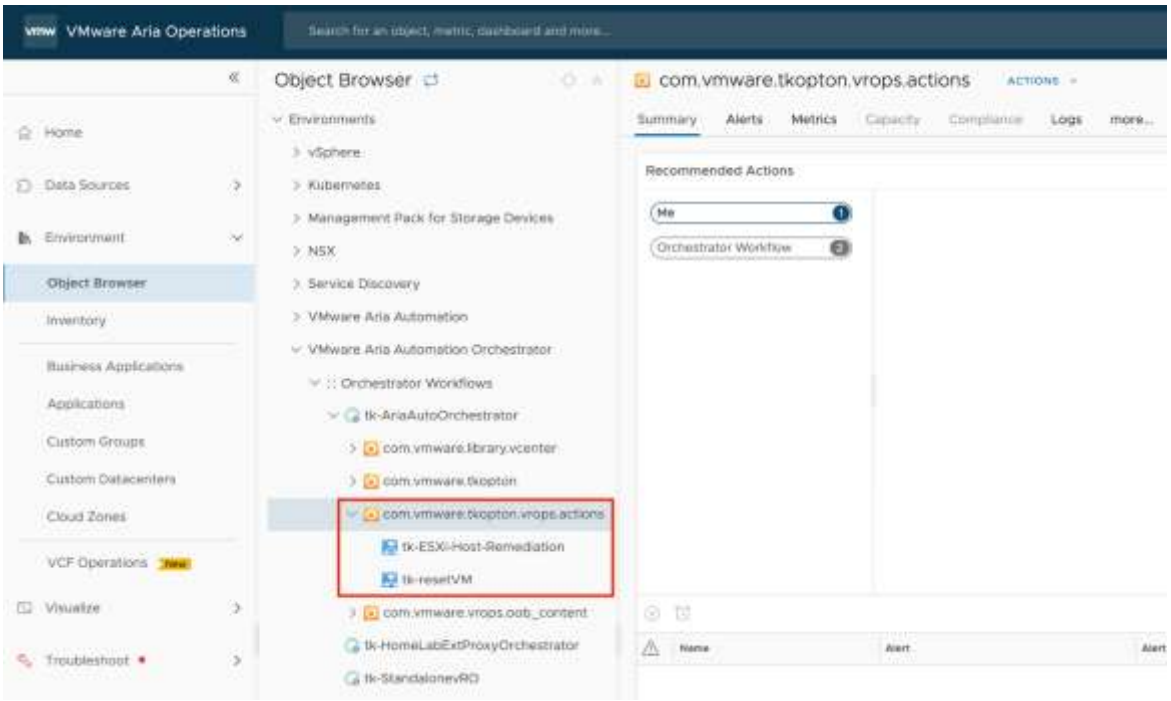


Figure 26: Discovered workflows.

If the use case necessitates executing the workflow on a specific vCenter object, such as a Virtual Machine in this example, it must be properly configured in Aria Operations. Aria Operations-driven automation needs to identify the resource types within an alert definition that will trigger the alert, and the resource type in Aria Automation Orchestrator on which the configured workflow can be executed. These two object types are usually the same but they do not have to be identical.

Furthermore, a workflow can have several different resource types for which it will be available. The target type is, of course, always the same, as it is the input expected by the workflow.

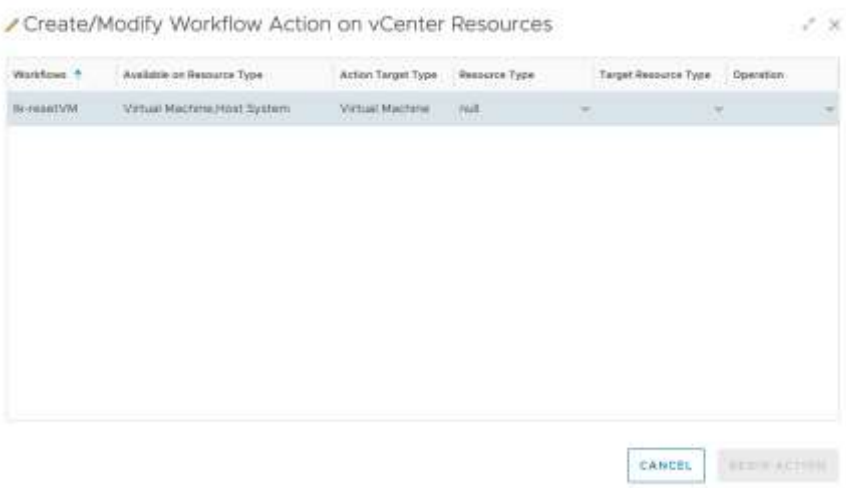


Figure 27: Workflow Action configuration with multiple Resource Types.

To configure this, run the “Create/Modify Workflow Action on vCenter Resources” from the “Actions” menu in the context of the specific workflow, as shown in the following figure.

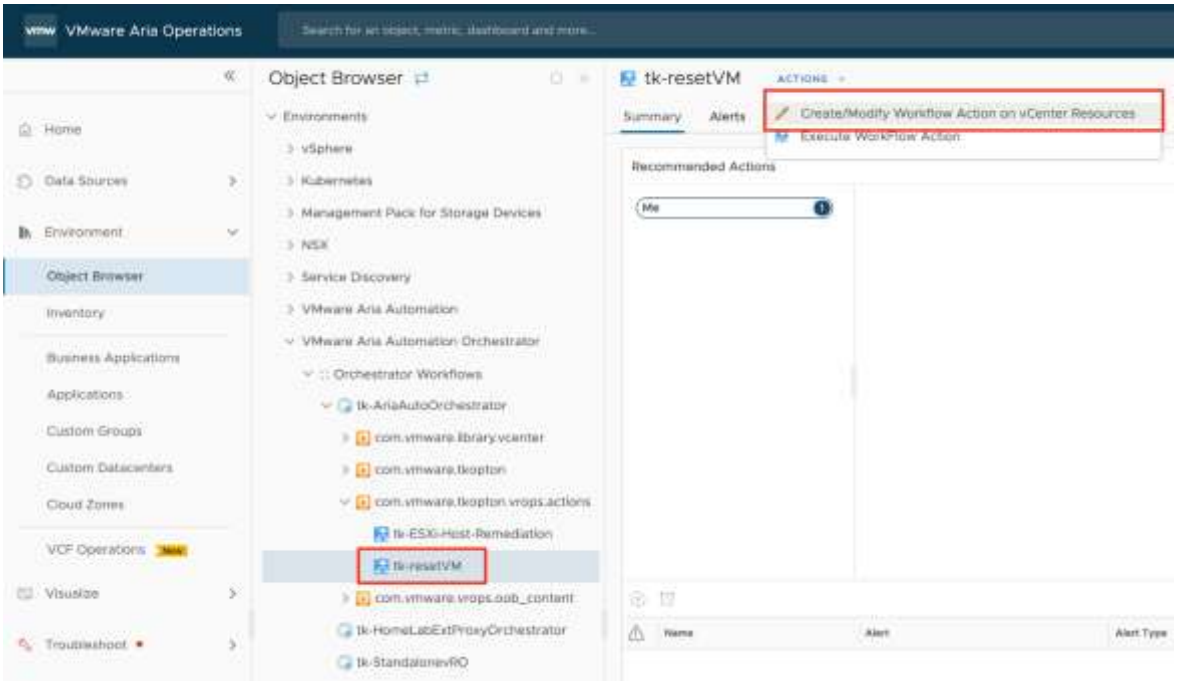


Figure 28: Configuring the workflow for specific resource type.

The parent object is the **Resource Type**, while the **Target Type** refers to the specific vCenter object for which the workflow is intended. The workflow takes the target object as its input parameter. For instance, if you want to map a workflow called “Change Ram” to a host system and run this workflow on multiple virtual machines under that host, you should select the Resource Type as Host System and the Target Type as Virtual Machine.

In the example use case, the alert will be triggered on a Virtual Machine resource type and executed on a Virtual Machine resource in vCenter. To achieve this, configure the Virtual Machine as shown in the following picture. The final step is to click “Add” to configure the workflow as a usable action.

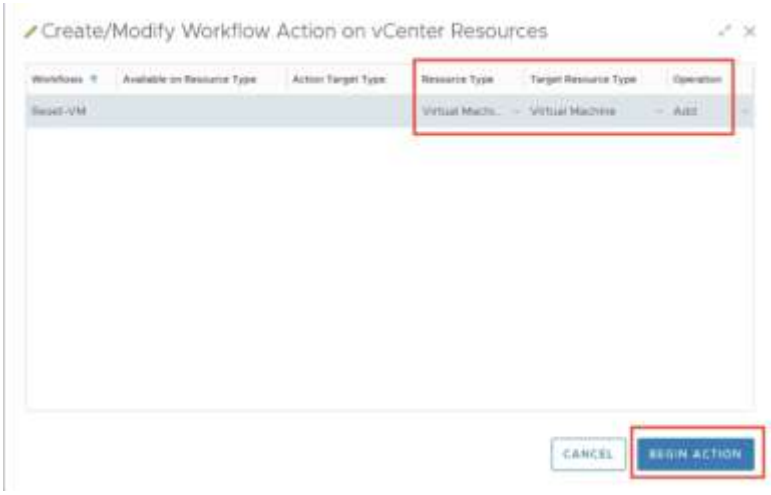


Figure 29: Workflow Action parameters configuration.

With this configuration, Aria Operations has all the necessary information to automate tasks using Aria Automation Orchestrator workflows. The final step is to configure these workflow-based actions within the appropriate Recommendation definition and enable the automation within the respective policy.

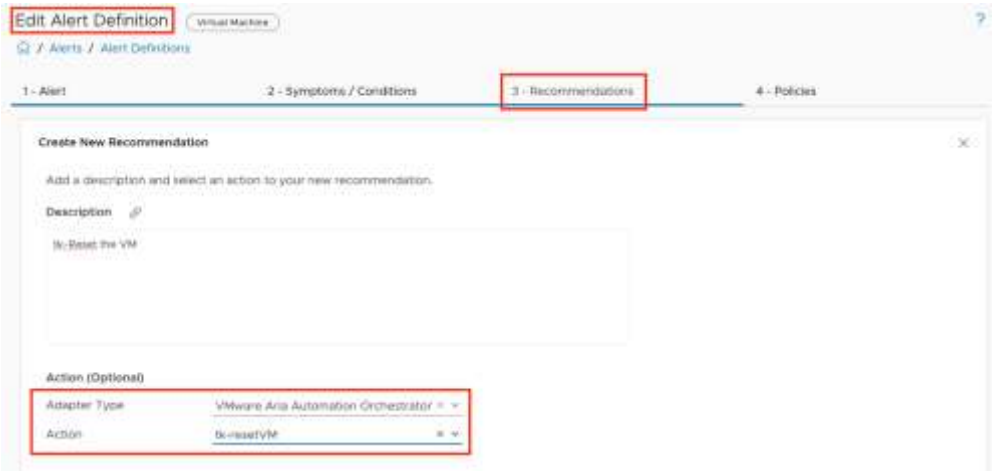


Figure 30: Alert Definition with a Recommendation containing a Workflow Action.

Upon execution, the properly configured Aria Automation Orchestrator workflow will receive the Aria Operations alert ID and a reference to the affected Virtual Machine, as shown in the next figure. This ID can be used to further enhance the automated actions.

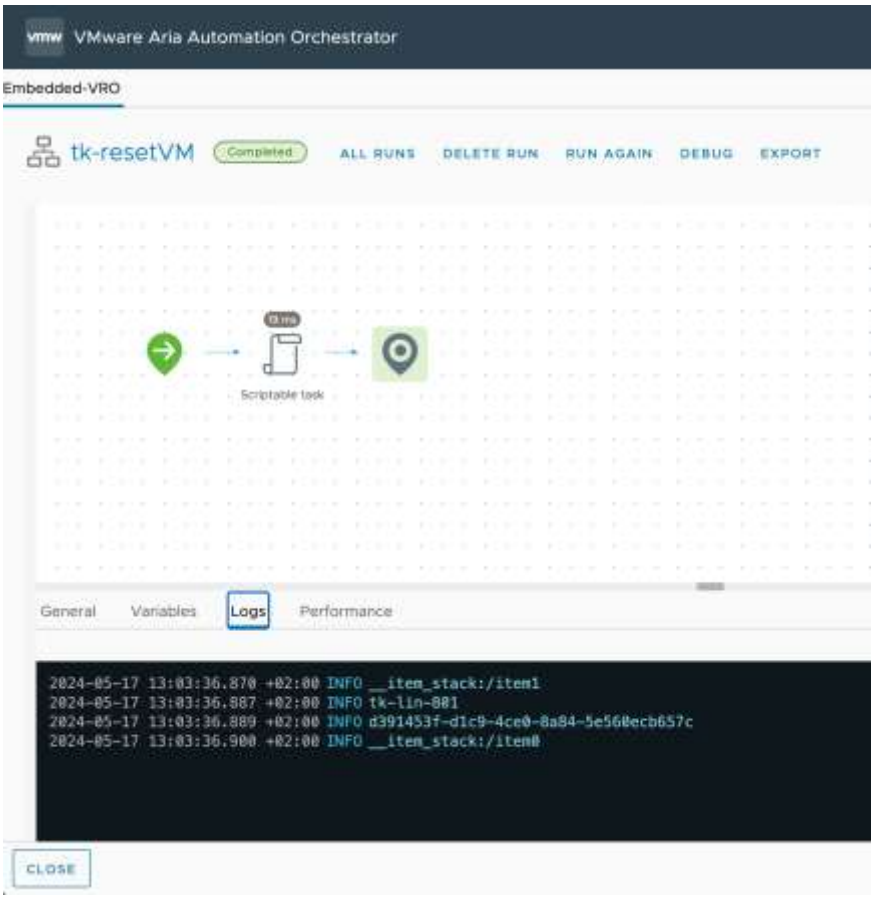


Figure 31: Workflow output in Aria Automation Orchestrator.

With the management pack for Aria Automation Orchestrator we have almost unlimited possibilities to extend Aria Operations actions and implement self-healing operations.

Automation Central

Automation Central is a powerful feature within Aria Operations that enables IT administrators to create, manage, and schedule automated jobs for optimizing virtual environments. This centralized hub allows users to automate various optimization actions, including reclaiming resources, rightsizing VMs, and performing routine maintenance tasks across vCenter and public cloud environments.

Automation Central serves as a cornerstone for efficient infrastructure management. It provides a user-friendly interface where administrators can set up recurring jobs, track their execution, and obtain detailed reports on the impact of these automated actions. By leveraging Automation Central, organizations can significantly reduce manual workload, improve resource utilization, and maintain a more optimized and cost-effective virtual infrastructure.

Key capabilities of Automation Central include scheduling actions like deleting old snapshots, powering off idle VMs, rightsizing oversized VMs, and even performing operations across multiple cloud platforms. This chapter will delve into the features, benefits, and best practices for utilizing Automation Central to streamline your VMware environment management.

Let's start with a look at the main page of Automation Central. Here we see at a glance all currently scheduled jobs in a calendar overview. As is typical in a calendar, we can navigate through the months, view the details of existing jobs, and start creating new jobs with just a click on "Add Job".

On the Automation Central Home Screen, we can also view **Reclamation** and **Rightsizing Reports**, which give us a quick overview of the outcome of our jobs in terms of optimized resources.

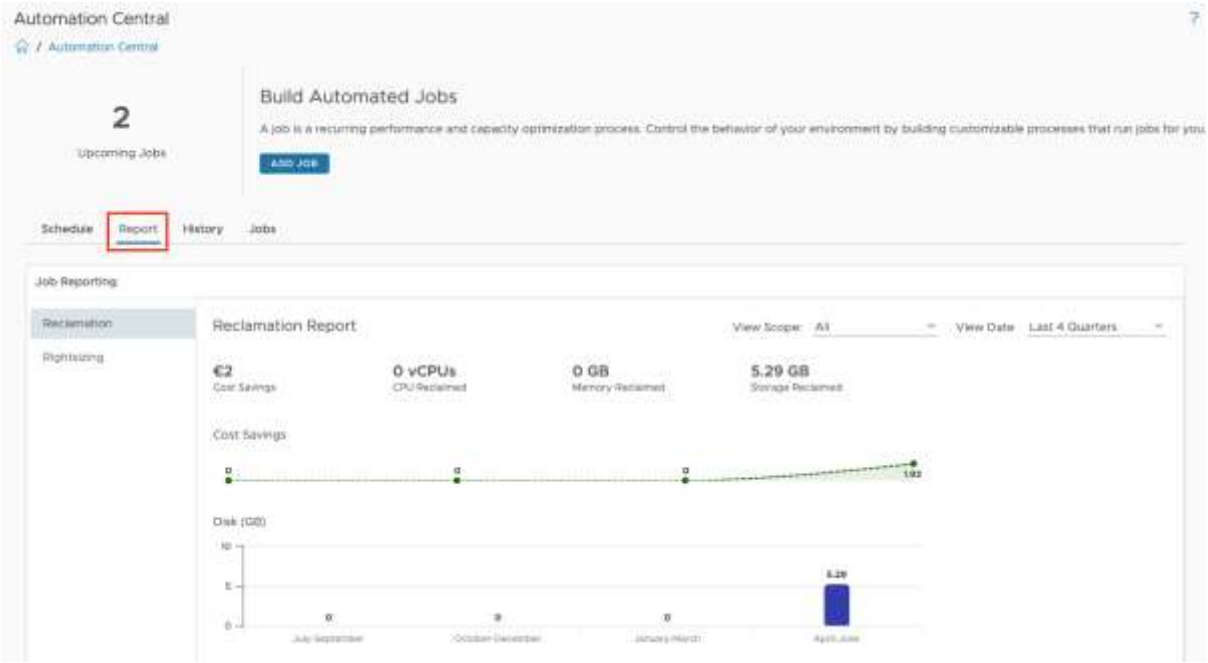


Figure 32: Automation Central - Reclamation Report

To review the **History** of configured jobs that have run, navigate to the History tab above the calendar. This tab provides a tabular view of job names and details. You can use the search function to find specific jobs or VMs, and the drop-down in the search box allows for advanced search options. Note that the job history page only displays the status of jobs.

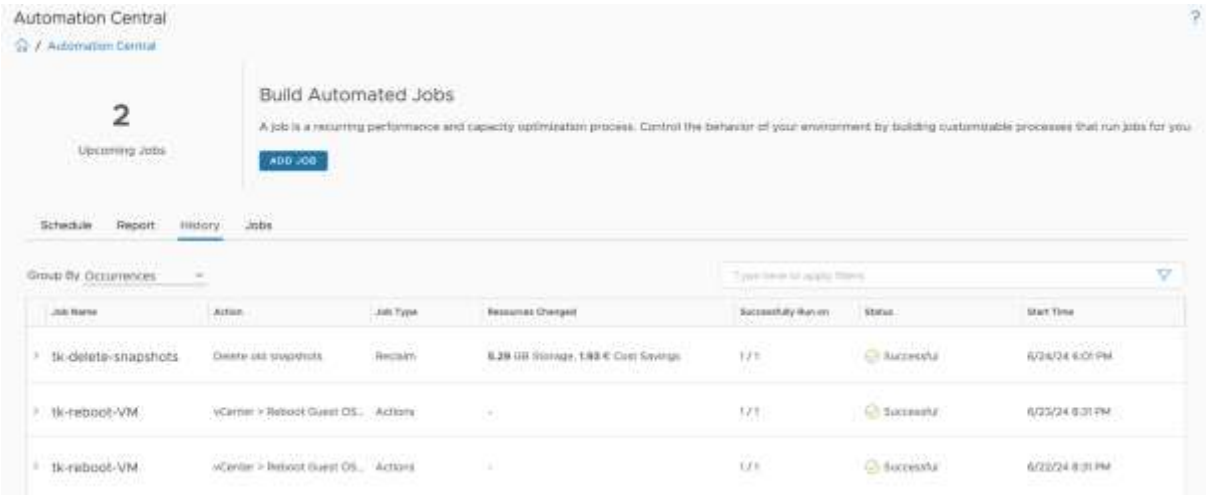


Figure 33: Automation Central - Job History

The **Jobs** tab displays a list of all configured jobs. For each job, clicking the ellipses icon brings up a menu where you can edit, delete, clone, or deactivate the job. If a job you created is not visible in the list, you can search for it by entering the name in the search box. Alternatively, you can check the job status by using the advanced search options available in the drop-down menu of the search box.

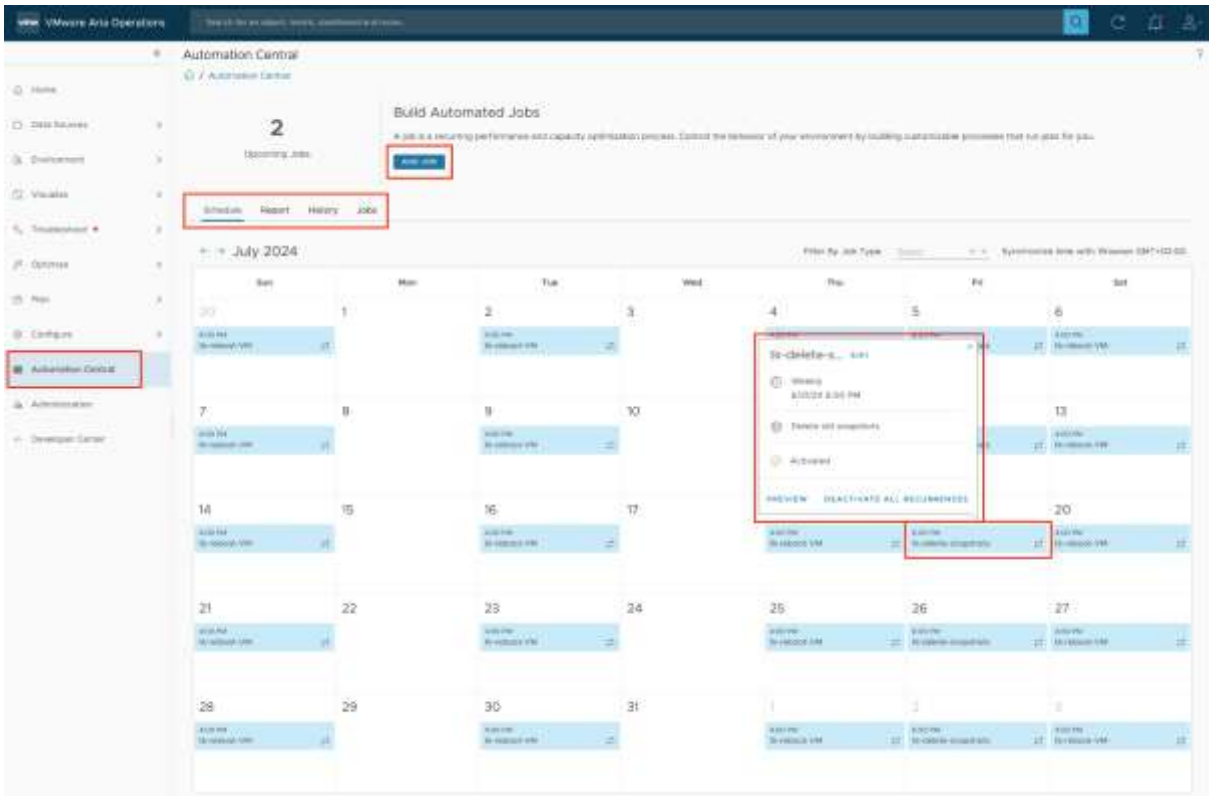


Figure 34: Automation Central - Schedule overview

Creating new fully automated jobs begins with selecting the action to be performed in the job. At the time of writing this chapter, the following three options are available:

- **Reclaim**: to optimize unused resources. This action may include one of the following options: Delete old snapshots, Delete idle VMs, Power off idle VMs, Delete powered off VMs.
- **Rightsize**: to optimize the configuration of VMs. This action may include one of the two options: Downsize oversized VMs or Scale-up undersized VMs
- **Actions**: to schedule reboots or power cycles in a multi-cloud environment including Public Cloud offerings.

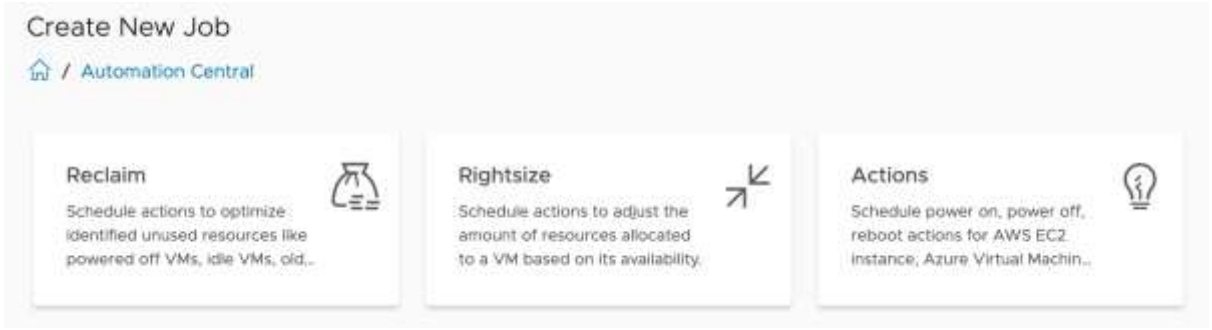


Figure 35: Automation Central - Three types of automated jobs

Creating an automated job is a **four-step** process. The following screenshots illustrate each step using a Rightsizing job as an example.

In the first step, "1-Rightsize Information", we define the **specific action** to be performed. In the case of Rightsizing, we have the choice between Downsize and Scale-Up. In this step, the parameters for the selected action are also specified.

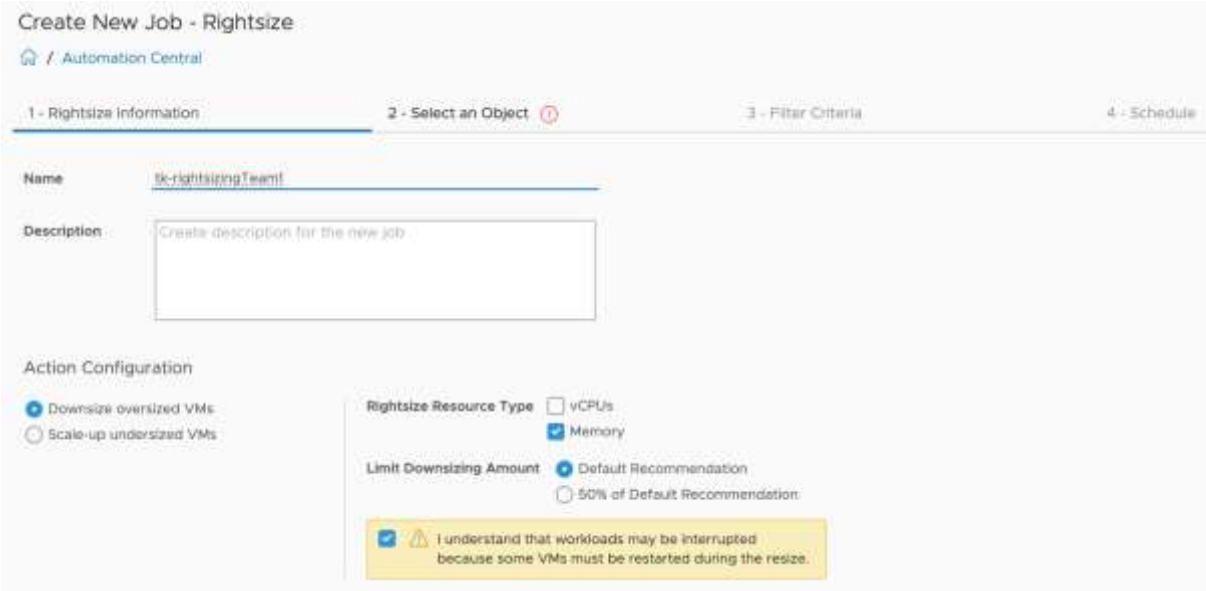


Figure 36: Automation Central - Creating a new job, step 1

In the second step, we define which objects the job should be executed on. Special caution is required here to ensure that the **scope** is not set too broadly, potentially affecting VMs that were not intended for automation.

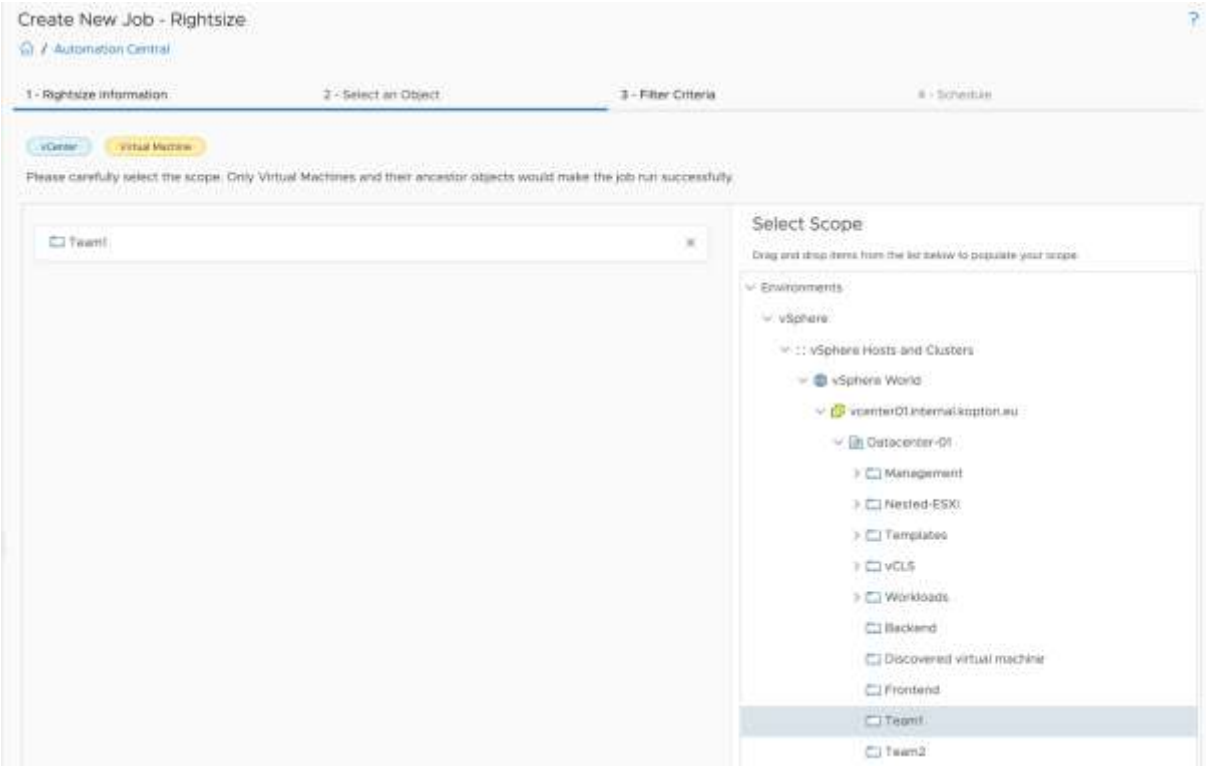


Figure 37: Automation Central - Creating a new job, step 2, selecting object(s)

In the third step, we can set **filters** to further refine the scope defined in the previous step. For example, a filter could be configured here to capture only VMs with a specific tag within the parent scope.



Figure 38: Automation Central - Creating a new job, step 3, filtering criteria

Finally, in the fourth step, the **schedule** is defined to determine when the job should be executed.

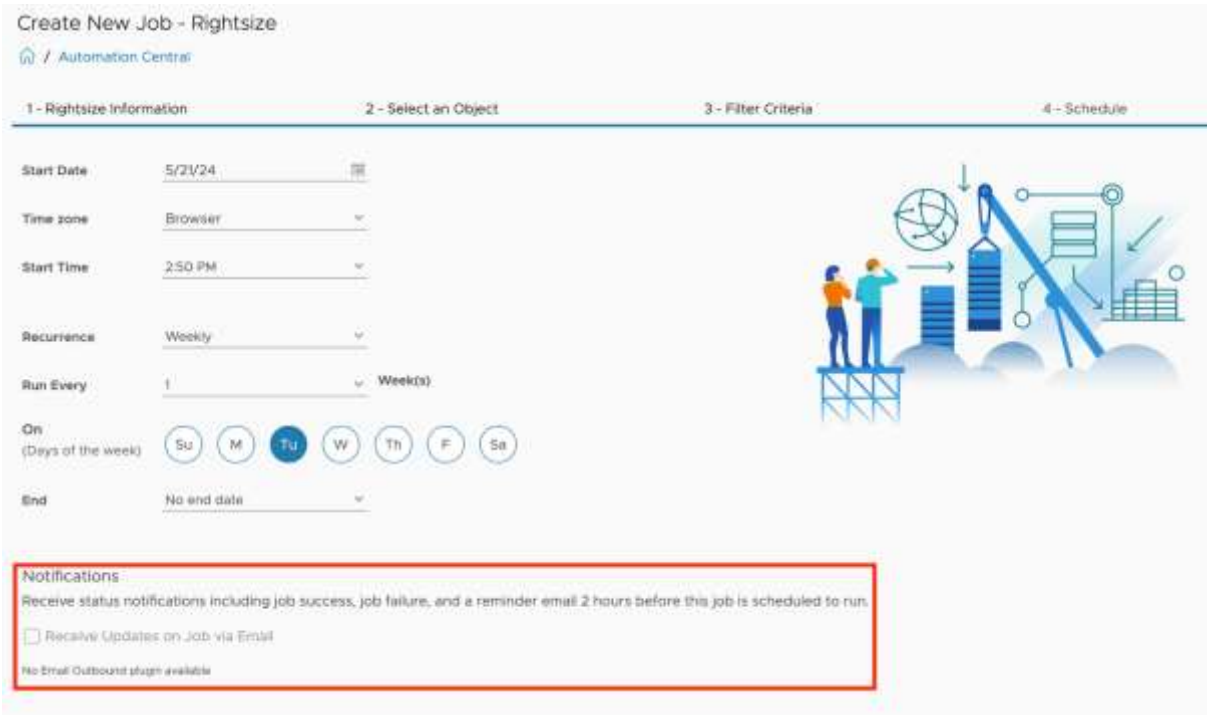


Figure 39: Automation Central - Creating a new job, final step 4, the schedule

While Automation Central provides a convenient way to schedule routine VM operations, its capabilities are currently limited compared to more advanced automation tools like Aria Automation Orchestrator. However, as already stated, it offers a simple and centralized interface within Aria Operations for common VM lifecycle management tasks.

Incoming Communication

Even though Aria Operations offers extensive possibilities with the Management Pack for Aria Automation Orchestrator and the Aria Operations Webhooks to control various systems and perform diverse automation tasks, these solutions are fundamentally 'fire-and-forget' or open-loop approaches as depicted in the following figure. This means that once an automation task is initiated, it runs to completion without ongoing feedback or adjustments based on real-time conditions.

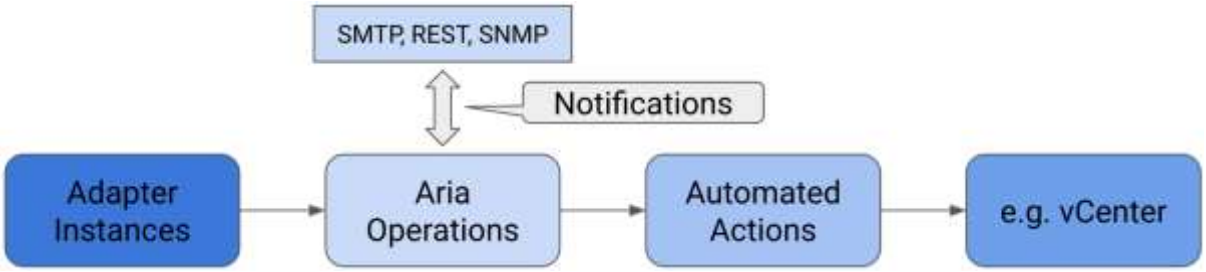


Figure 40: Open Control Loop

Naturally, we can design the primary workflow to execute other workflows and integrate additional systems to enhance the overall automation process. However, the question arises: how do we close the loop back to Aria Operations? What mechanisms are available to return to Aria Operations and obtain further information? Is it possible to extend Aria Operations with custom object types and metrics? The answer is straightforward: yes, it is possible.

By leveraging the Application Programming Interface (API), we can create a feedback loop that allows automated tasks to receive data from Aria Operations and make informed decisions based on the latest information. This capability enables more dynamic and responsive automation, ensuring that the system can adapt to changing conditions.

REST API

Aria Operations, in line with contemporary enterprise solutions, provides a comprehensive REST API for programmatic interaction with the platform. Since the release of version 8.2, VMware has enhanced accessibility to the REST API documentation by integrating Swagger UI. This interactive documentation is now available on every Aria Operations instance, offering developers and administrators direct access to API specifications and testing capabilities.

The REST API documentation can be accessed through the following URL on any Aria Operations deployment:

```
https://<AriaOps FQDN>/suite-api/doc/swagger-ui.html
```

This integration of Swagger UI not only simplifies API exploration but also promotes more efficient development and integration processes, allowing users to interact with the API directly from the documentation interface.

The next image shows the landing page of the **Suite API Swagger** documentation, which offers very simple navigation and an overview of the available methods.

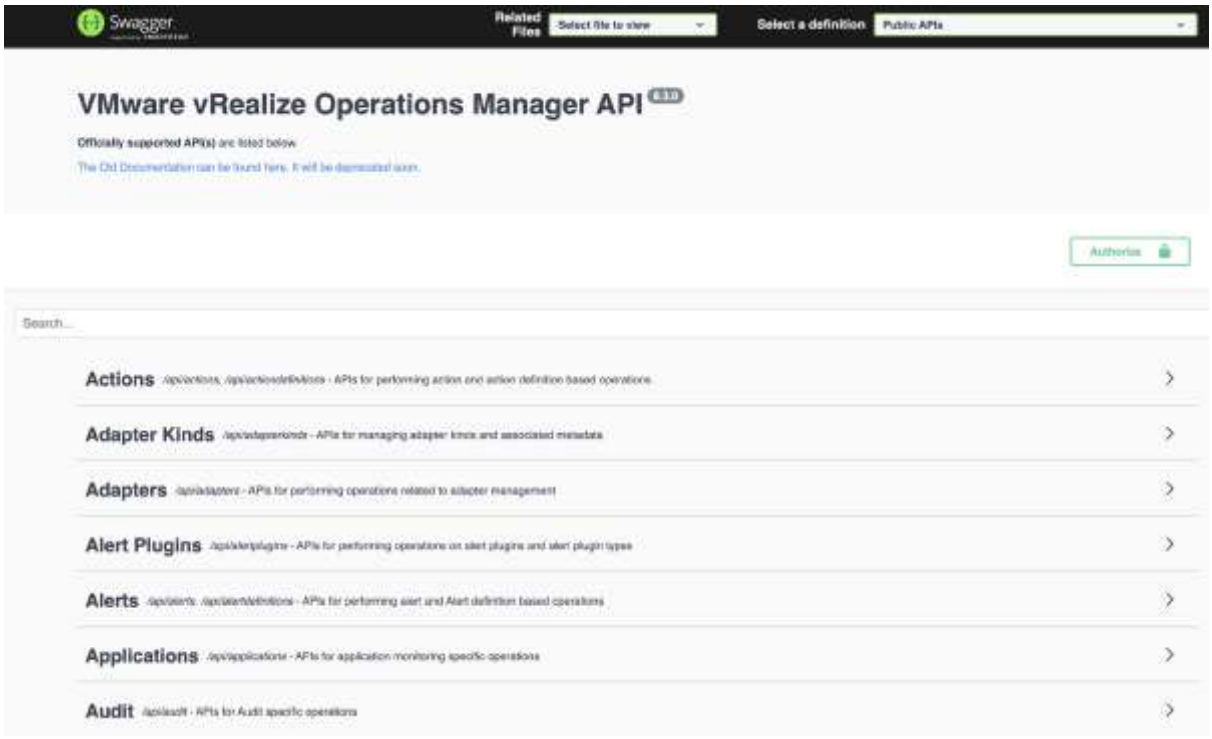


Figure 41: Aria Operations Swagger UI Suite-API documentation.

The Aria Operations REST API provides comprehensive programmatic access to all entities managed by the platform, including their associated metrics, properties, and Aria Operations constructs such as Alert and Symptom Definitions and Recommendations. With appropriate authorization, users can not only extract data from Aria Operations but also introduce new information into the system.

This capability extends to the creation of entirely new custom objects, either of existing or new types, complete with their unique sets of metrics and properties. Furthermore, users can enhance existing objects already collected by Aria Operations adapter instances with custom metrics and properties.

The API offers methods to programmatically manipulate, create, or delete various Aria Operations constructs, including but not limited to Alert and Symptom Definitions, Adapter Configurations, and Super Metrics.

As of the current release, the API supports both XML and JSON formats for data exchange, with JSON being the preferred and recommended format due to its efficiency and widespread adoption in modern web services.

This robust API framework enables developers and administrators to integrate Aria Operations seamlessly with other systems, automate complex workflows, and extend the platform's capabilities to meet specific organizational requirements.

Swagger UI provides an intuitive interface for testing available API methods before implementing them in code. Whether utilizing Swagger UI or working directly within the codebase, the initial step involves authentication. In the context of programmatic access, Aria Operations requires the use of the 'acquire' POST method to obtain an object containing an authentication token and its associated validity period. This token is then utilized in subsequent REST API calls to maintain a secure session.

The authentication process can be visualized and tested through Swagger UI, which offers a user-friendly representation of the 'acquire token' method. This interactive documentation not only facilitates understanding of the API structure but also allows for real-time testing of API endpoints, ensuring developers can validate their authentication approach before full implementation.

Figure 42 illustrates the 'acquire token' method as presented in Swagger UI, demonstrating the required parameters and expected responses for successful authentication.

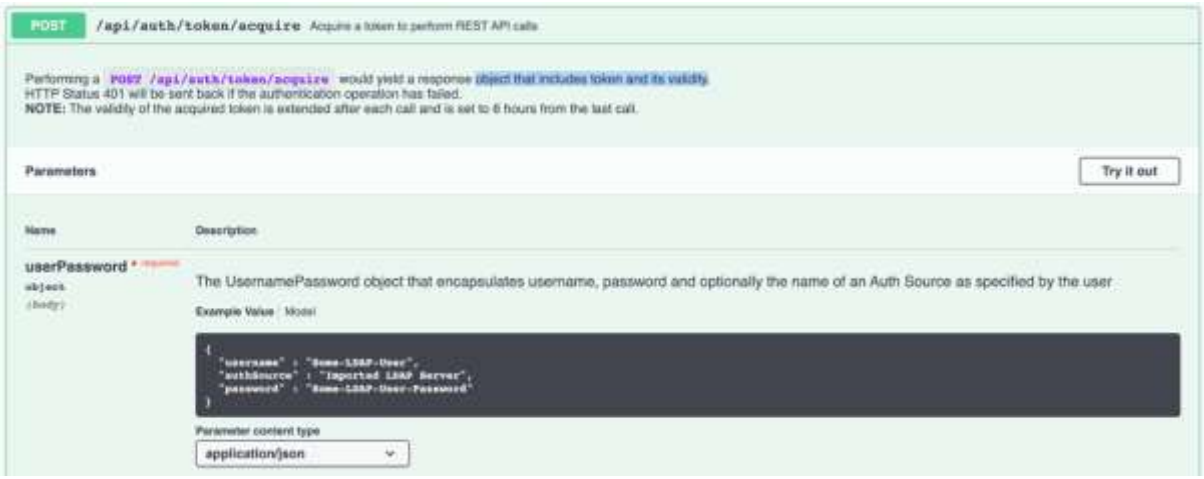


Figure 42: Acquire token REST API call.

This approach to API exploration and authentication setup exemplifies the robust and developer-friendly nature of Aria Operations' API infrastructure, enabling efficient integration and custom solution development.

The Swagger UI provides an efficient mechanism for identifying the appropriate API function. Users can utilize the search functionality by entering relevant keywords, which then filters and displays the available options. Figure 43 illustrates a curated list (shortened) of accessible methods pertaining to Aria Operations Alerts, demonstrating the intuitive nature of the Swagger interface.

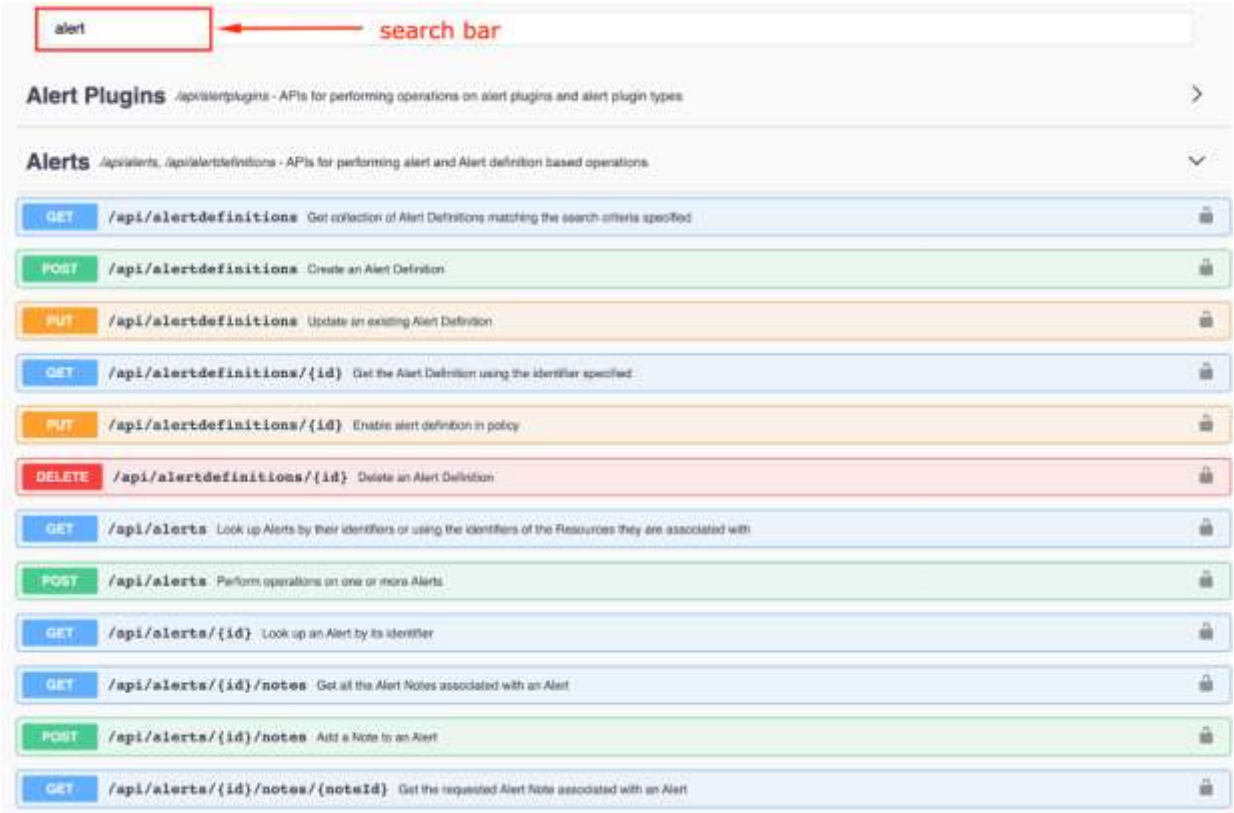


Figure 43: Finding method in the Swagger UI documentation.

This search-driven approach to API exploration not only streamlines the development process but also enhances the overall user experience, allowing developers to quickly locate and understand the specific endpoints required for their integration or automation tasks. The Swagger UI's interactive documentation serves as a valuable resource, bridging the gap between API specification and practical implementation in the context of Aria Operations.

The Swagger UI provides a straightforward method for identifying the appropriate function. By simply entering your search term, you can observe the available options. The following illustration presents a condensed list of accessible methods related to Aria Operations Alerts.

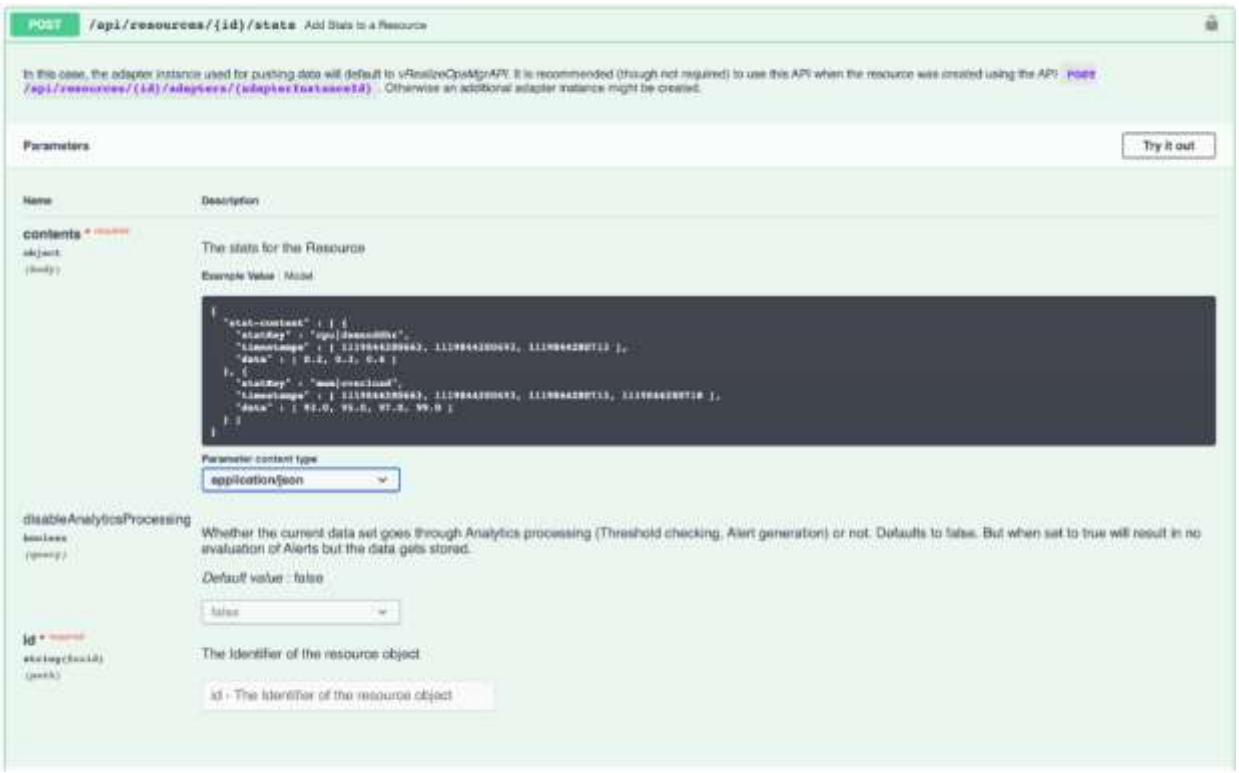


Figure 44: Add Stats REST POST call example.

In a concise example, we will examine the structure of a JSON payload, also referred to as the body, required to append a new metric and a new property to an existing object, in this case, an ESXi host, via the API.

```
{
   "stat-content": [
       {
           "statKey": "CustomMetrics|Fan01|Speed",
           "timestamps": [
               {{epoch}}
           ],
           "data": [
               1700.0
           ],
           "others": [],
           "otherAttributes": {}
       }
   ]
}
```

Please note that this example was executed in Postman, utilizing an *{{epoch}}* variable for determining the current timestamp. This variable contains the result of a pre-request script executed at runtime.

The script code in Postman:

```
var epoch = (new Date).getTime();
postman.setEnvironmentVariable("epoch", epoch);
```

The corresponding curl command (FQDN, ObjectID and the Token have been shortened):

```
curl --location 'https://vrops/suite-api/api/resources/18xxx2/stats' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: vRealizeOpsToken f0xxx1' \
--data '{
    "stat-content": [
        {
            "statKey": "CustomMetrics|Fan01|Speed",
            "timestamps": [
                1720088952741
            ],
            "data": [
                1700.0
            ],
            "others": [],
            "otherAttributes": {}
        }
    ]
}'
```

And this is how the result looks like in the Aria Operations UI.

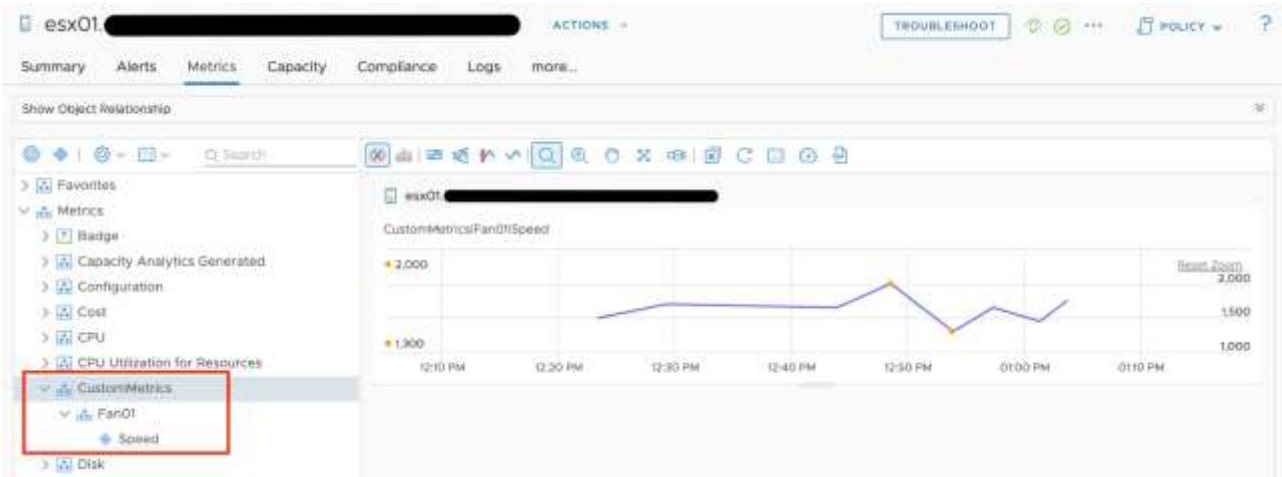


Figure 45: Custom Metrics in Aria Operations UI

Example of a JSON body showing how to add a custom property:

```
{
   "property-content": [
       {
           "statKey": "CustomProps|Fan01|Status",
           "timestamps": [
               {{epoch}}
           ],
```

```
            "values": [
                "ON"
            ],
            "others": [],
            "otherAttributes": {}
        }
    ]
}
```

The corresponding curl command (FQDN, ObjectID and the Token have been shortened):

```
curl --location 'https://vrops/suite-api/api/resources/18xxx2/properties' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: vRealizeOpsToken f0xxx1' \
--data '{
    "property-content": [
        {
            "statKey": "CustomProps|Fan01|Status",
            "timestamps": [
                1720099341117
            ],
            "values": [
                "ON"
            ],
            "others": [],
            "otherAttributes": {}
        }
    ]
}'
```

And the representation in the UI:

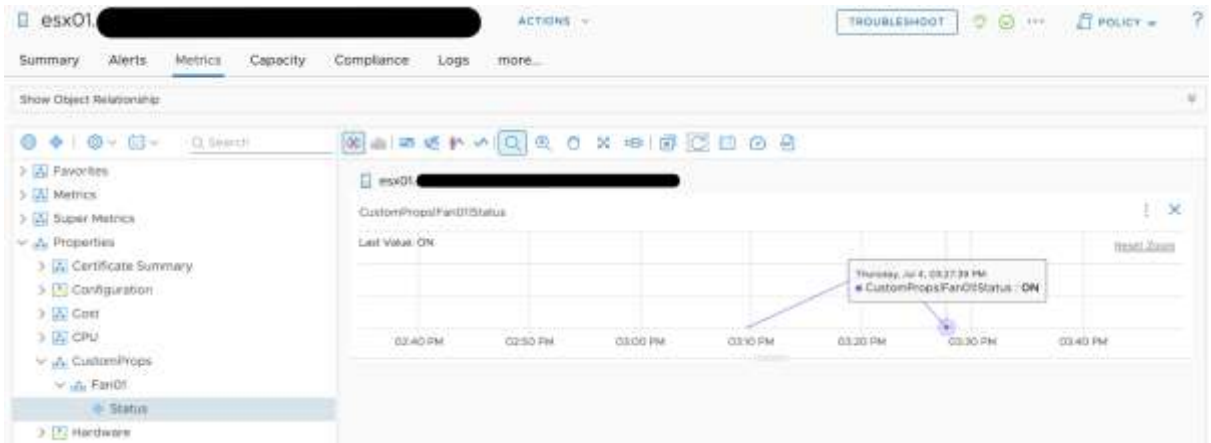


Figure 46: Custom Property in Aria Operations UI

The available API methods are divided into two distinct groups: the public API definitions and the internal API.

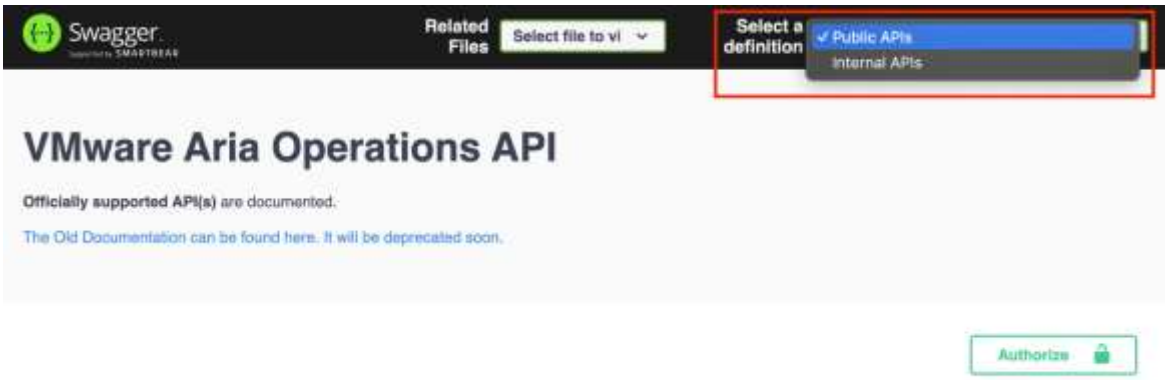


Figure 47: Public and Internal API

While the public API is considered stable and supported, the internal API methods should be used with caution. Please note that the internal API methods may be subject to change or removal in future releases without prior notice. Utilizing the internal API is done at the user's own risk.

One remarkable use case leveraging the Aria Operations REST API is the vRealize Operations Export Tool (vrops-export), developed by Pontus Rydin. This utility facilitates the export of data from Aria Operations.

The latest version, as of the time of writing these words, can be found here.

Closing the Loop - Callbacks

With the ability to connect back to Aria Operations to retrieve data or push custom information, we can finally close the loop to achieve a fully featured closed-control loop. The following diagram represents our closed-loop control system, incorporating all concepts we have learned in the previous sections of this chapter.

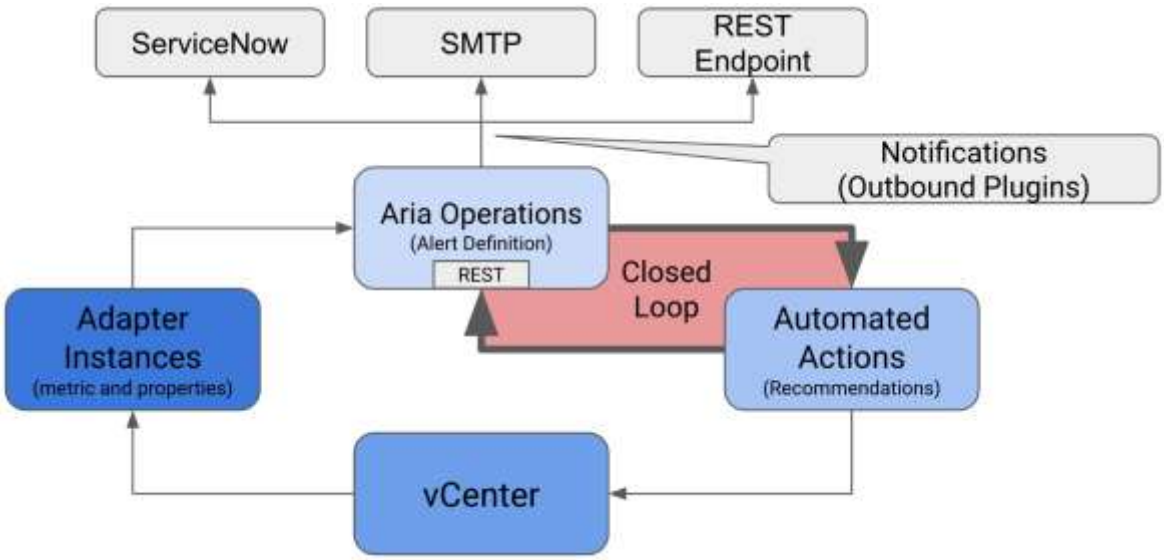


Figure 48: Closed Control Loop

Upon comparing the diagram to Figure 2 “Open Control Loop”, you will notice that we now have a continuous and closed-loop between the entity executing our automated actions and the entity collecting the data that reflect the behavior of objects we control. We are no longer limited to the "fire and forget, until the sensors do something" method. With the possibility to execute callbacks to the REST API, Aria Operations becomes an integral part of an automated SDDC solution.

Let us revisit our initial use case: "If a VM (the OS) crashes, this VM should be hard-reset." We can expand this use case to achieve a sophisticated and automated remediation process leveraging all the concepts presented in this chapter. Let us revisit the diagram xy and extract the possible components of the automation.

The central control point is, of course, Aria Operations itself. This platform provides the necessary tools and integrations to automate and manage VMs and other objects effectively.

Within Aria Operations, we have an Alert Definition that leverages specific symptoms to trigger an alarm when the symptoms indicate that something is amiss, such as a VM crash. This alarm is based on symptom evaluation, which may not directly point to a root cause.

Upon alarm generation, the configured Notification:

- Creates a problem ticket in ServiceNow.
- Sends a notification email to the admin team.

Simultaneously, the Action configured within the Recommendation initiates a Aria Automation Orchestrator Workflow. This workflow may:

- Attempt to connect to the VM via RDP or SSH.
- Execute ping or TCP connect checks.
- Reset the VM if the configured checks support the initial assumption.
- Verify the VM's availability.
- Update the alert status in Aria Operations if necessary.
- Push additional properties (e.g., count of Aria Automation Orchestrator-initiated reset events) to the VM object.

When the alarm status changes, the configured Notification:

- Updates the problem ticket in ServiceNow.
- Sends another notification email to the admin team.

With Aria Operations, its comprehensive REST API, the wide range of Management Packs, integrated Notification Plugins, and the capability to run Actions using Aria Automation Orchestrator, you have extensive possibilities to automate your SDDC management and elevate it to become a Self-Driving SDDC.

SDDC vs IaaS

Part 4 Chapter 3

What you architected is SDDC, but what you operate is IaaS. You buy system, you sell service.

This chapter provides an introduction to how IaaS management differ to the physical infrastructure management. It is placed at the end as this is now considered a basic knowledge. I still find it useful in explaining why the complexity goes up significantly. IDC actually predicts this back in 2012. The link from APM Digest website states that “operational complexity in virtualized, scale-out, and cloud environments and composite Web-based applications will drive demand” for a new type of tools that can “rapidly sort through hundreds of thousands of monitor variables, alerts and events”.

VM, it is not what you think!

In this chapter, we will dive into why seemingly simple technology, an X86 machine virtualized, has a large ramification for the IT industry. In fact, it is turning a lot of things upside down and breaking down silos that have existed for decades in large IT organizations. We will cover the following topics:

- Why virtualization is not what we think it is
- Virtualization vs Partitioning
- A comparison between a physical server and a virtual machine

Journey into the Virtual World

It is the era of the cloud. Who does not know what a VM is? Even a business user who has never seen one knows what it is. It is just a physical server, virtualized. Nothing more.

Wise men say that small leaks sink the ship. We think that's a good way to explain why IT departments who manage physical servers well struggle when the same servers were virtualized.

We can also use Pareto principle (80/20) rule. 80% of a VM is identical to physical server. But it's the 20% differences that hit you. We will highlight some of this "20% portion", focusing on areas that impact data center management.

The change caused by virtualization is much larger than the changes brought forward by previous technologies. In the past two or more decades, we transitioned from mainframes to the client/server-based model to the web-based model. These are commonly agreed upon as the main evolutions in IT architecture. However, all of these are just technological changes. It changes the architecture, yes, but it does not change the operation in a fundamental way. Both the client-server and web shifts did not talk about the "journey". There was no journey to the client-server based model. However, with virtualization, we talk about the virtualization journey. It is a journey because the changes are massive and involve a lot of people. That's why the evolution toward multi-cloud operations is also called a journey.

Gartner correctly predicted the impact of virtualization in 2007. Two decades later we still have not completed the journey. Proving how pervasive the change is, here is the following summary on the article[1] from Gartner:

[1] The link from Gartner no longer works, so I guess you gotta trust me now 😉



Notice how Gartner talks about change in culture. So, virtualization has a cultural impact too. In fact, if your virtualization journey is not fast enough, look at your organization's structure and culture. Have you broken the silos? Do you empower your people to take risk and do things that have never been done before? Are you willing to flatten the organization chart?

The siloes that have served you well is likely your #1 barrier to multi-cloud. The complexity of multi-cloud have made some customers repatriating workload back to private cloud.

So why exactly is virtualization causing such a fundamental shift? To understand this, we need to go back to the basics, which is what exactly virtualization is. It's pretty common that senior IT management have a misconception about what this actually is.

Take a look at the following comments. Have you seen them in your organization?

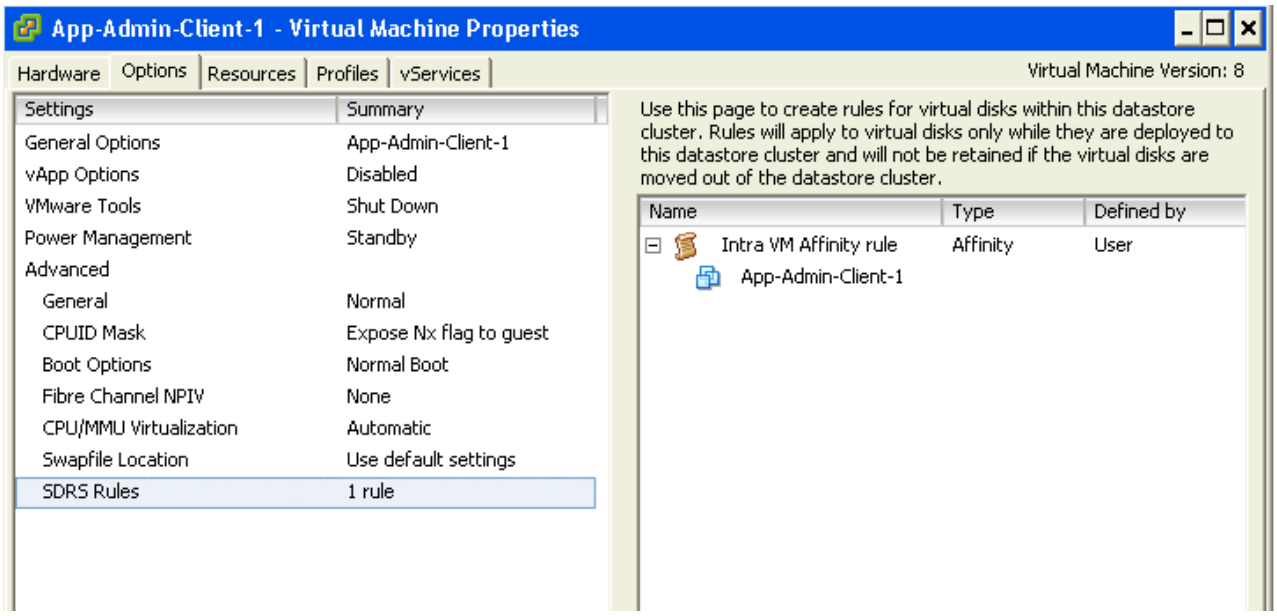
- "VM is just Physical Machine virtualized. Even VMware said the Guest OS is not aware it's virtualized and it does not run differently."
- "It is still about monitoring CPU, RAM, Disk, Network. No difference."
- "It is a technology change. Our management process does not have to change."
- "All of these VMs must still feed into our main Enterprise IT Management system. This is how we have run our business for decades and it works."

If only life was that simple, we would all be 100 percent virtualized and have no headaches! Virtualization has been around for decades, and yet most organizations have not mastered it. The proof of mastering if you have completed the journey and have reached the highest level of virtualization maturity model.

Although virtualization looks similar on the cover to a physical world, it is completely re-architected *under the hood*.

Virtual Machine vs Physical Machine

VM is not just a physical server virtualized. Yes, there is a P2V process. However, once it is virtualized, it takes on a new shape. That shape has many new and changed properties, and some old properties are no longer applicable or available. The following is an old screenshot, taken years ago. Can you spot properties that do not exist in physical server?



Let's highlight some of the properties that do not exist in a physical server. I'll focus on those properties that have an impact on management, as management is the topic of this book.

Properties	Physical Server	VM
BIOS	A unique BIOS for every brand and model. Even the same model (for example, HP DL 380 Generation 9) can have multiple versions of BIOS. BIOS needs updates and management, often with physical access to a data center. This requires downtime.	This is standardized in a VM. There is only one type, which is the VMware motherboard. This is independent from the ESXi motherboard. VM BIOS needs far less updates and management. The inventory management system no longer needs the BIOS management module.
Virtual HW	Not applicable	This is a new layer below BIOS. It needs an update on every vSphere release. A data center management system needs to be aware of this as it requires a deep knowledge of vSphere. For example, to upgrade the Virtual Hardware, the VM has to be in the power-off stage.
Drivers	Many drivers are loaded and bundled with the OS. Often, you need to get from respective hardware vendors for the latest drivers. All these drivers need to be managed. This can be complex operation, as they vary from model to model and brand to	Relatively fewer drivers are loaded with the Guest OS; some drivers are replaced by the ones provided by VMware Tools. Even with NPIV, the VM does not need the FC HBA driver. VMware Tools needs to be managed, with vCenter being the most common management tool.

	brand. The management tool has rich functionalities, such as checking compatibility, rolling out drivers, rolling back if there is an issue, and so on.

With all the above differences, how does it impact the hardware upgrade process?

Physical Server	VM
Downtime required. It is done offline and is complex. OS reinstallation and updates are required, hence it is a complex project in the physical world. Sometimes, a hardware upgrade is not even possible without upgrading the application.	It is done online and is simple. Virtualization decouples the application from hardware dependency. A VM can be upgraded from a 5-year-old hardware to a new one, moving from the local SCSI disk to 40 Gb FCoE, from dual core to a 24-core CPU. So yes, MS-DOS can run on 100 Gb Ethernet accessing SSD storage via the PCIe lane. You just need to perform vMotion to the new hardware. As a result, the operation is drastically simplified.

Storage

In the preceding table, we compared the core properties of a physical server with a VM. Every server needs storage, so let's compare the storage properties.

Physical Server	VM
For servers connected to SAN, they can see the SAN and FC fabric. They need HBA drivers and have FC PCI cards, and have multipathing software installed. Normally needs an advanced filesystem or volume manager to RAID local disk.	No VM is connected to FC fabric or the SAN. VM only sees the local disk. Even with N_Port ID Virtualization (NPIV) and physical RDM, the VM does not send FC frames. Multipathing is provided by vSphere, transparent to VM. There is no need for RAID local disk. It is one virtual disk, not two. Availability is provided at the hardware layer.
Backup agent and backup LAN are needed in the majority of cases.	They are not needed in the majority of cases, as backup is done via vSphere VADP API. Agent is only required for application-level backup.

Network and Security

Big difference in storage. How about Network and Security?

In vSphere, a VM is connected to a distributed virtual switch. It is not directly connected to the physical NIC in your ESXi host. The ESXi host's physical NICs become the virtual switch's uplinks instead. This means that the traditional top-of-rack (TOR) switch has been entirely virtualized. It runs completely as software. This means the management software needs to understand the distributed vSwitch and its features.

Physical Server	VM
NIC teaming is common. Typically needs two cables per server.	NIC teaming provided by ESXi. VM is not aware and only sees one vNIC.
Guest OS is VLAN aware. It is configured inside the OS. Moving VLAN requires reconfiguration.	VLAN is generally provided by vSphere, and not done inside the Guest OS. This means VM can be moved from one VLAN to another with no downtime. With network virtualization, VM is moving from VLAN to VXLAN.
The AV agent is installed on Guest, and can be seen by the attacker.	An AV agent runs on the ESXi host as a VM (one per ESXi). It cannot be seen by the attacker from inside the Guest OS.
AV consumes OS resources. AV signature updates cause high storage throughput.	AV consumes minimal Guest OS resources as it is offloaded to the ESXi Agent VM. AV signature updates do not require high IOPS inside the Guest OS. The total IOPS is also lower at the ESXi host level as it is not done per VM.

Management

Lastly, let's take a look at the impact on management. As can be seen next, even the way we manage a server changes once it is converted into a VMs

Property	Physical Server	VM
Approach on Monitoring	An agent is commonly deployed. It is typical for a server to have multiple agents. In-Guest metrics are accurate as the OS can see the physical hardware. A physical server has an average of 5 percent CPU utilization due to the multicore chip. As a result, there is no need to monitor it closely.	An agent is typically not deployed. Certain areas such as application and Guest OS monitoring are still best served by an agent. The key in-Guest metrics are not accurate as Guest OS does not see the physical hardware. A VM has an average of 50 percent CPU utilization as it is right sized. This is 10 times higher when compared with a physical server. As a result, there is a need to monitor closely, especially when physical resources are oversubscribed. Capacity management becomes a discipline in itself.
Approach on Availability	HA is provided by clusterware such as Microsoft Windows Server Failover Clusters (WSFC) and Veritas Cluster Server (VCS). Clusterware tends to be complex and expensive. Cloning a physical server is a complex task and requires the boot drive to be on the SAN or LAN, which is not typical.	HA is a built-in core component of vSphere. From what we see, most clustered physical servers end up as just a single VM as vSphere HA is good enough. Cloning can be done easily. It can even be done live. The drawback is that the clone becomes a new area of management.

	Snapshot is rarely done, due to cost and complexity. We find only very large IT departments practice physical server snapshot.	Snapshot can be done easily. In fact, this is done every time as part of backup process. Snapshot also becomes a new area of management as they tend to be forgotten.
Company Asset	The physical server is a company asset and it has book value in the accounting system. It needs proper asset management as components vary among servers. Here, the annual stock-take process is required.	VM is not an asset as it has no accounting value. A VM is like a document. It is technically a folder with files in it. Stock-take process is no longer required as the VM cannot exist outside vSphere.

SDDC vs HDDC

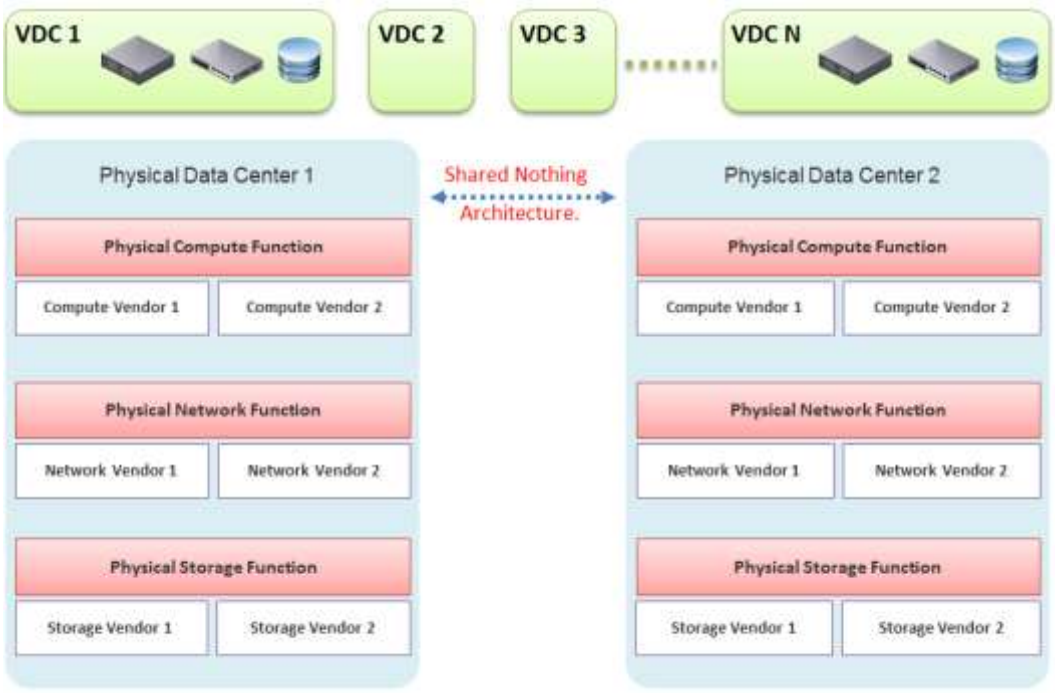
We covered how a VM differs drastically to a physical server. Now let's take a look at the big picture, which is at the data center level. A data center consists of three major functions—compute, network, and storage. Security is not a function on its own, but a key property that each function has to deliver. We use the term compute to represent processing power namely CPU and Memory. In today's data centers Compute is also used when referencing converged infrastructure, where the server and storage have physically converged into one box. The industry term for this is Hyper-Converged Infrastructure (HCI). You will see later in the book that this convergence impacts how you architect and operate SDDC.

VMware has moved to virtualize the network and storage functions as well, resulting in a data center that is fully virtualized and thus defined in the software. The software is the data center. This has resulted in the term SDDC. The book will make extensive comparison with the physical data center. For ease of reference, let's call the physical data center Hardware-Defined Data Center (HDDC).

In SDDC, we no longer define the architecture in the physical layer. The physical layer is just there to provide resources. These resources are not aware of one another. The stickiness is reduced, and they become a commodity. In many cases, the hardware can even be replaced without incurring downtime to the VMs running on top.

The next diagram shows one possibility of a data center that is defined in the software. We have drawn the diagram to state a point, so don't take this as the best practice for SDDC architecture. In the diagram, there are many virtual data centers (we draw three due to space in the book). Each virtual data center has its own set of virtual infrastructure (server, storage, network and security). They are independent of one another.

A virtual data center is no longer contained in a single building bound by a physical boundary. Although long distance WAN bandwidth and latency are still limiting factors in 2021, the main thing here is you can architect your physical data centers as one or more logical data centers. You should be able to automatically, with just a few clicks in VMware Site Recovery Manager, move thousands of servers from data center A to data center B; alternatively, you can perform DR from four branch sites to a common HQ data center.



In our example, the virtual data centers run on top of two physical data centers. Large enterprises will probably have more than that (whether it is outsourced or not is a different matter). The two physical data centers are completely independent. Their hardware is not dependent on one another.

- In the Compute function, there is no stretched cluster between 2 physical sites. Each site has its own vCenter. There is no need to protect vCenter with DR.
- In the Network function, there is no stretched VLAN between 2 physical sites. You do not have to worry about spanning tree or broadcast storm hitting multiple data centers. The physical sites can even be on a different network. Site 1 might be 10.10.x.x network, while Site 2 might be 20.20.x.x.
- In the Storage function, there is no array-based replication. Replication can be done independently from a storage protocol (FC, iSCSI, or NFS) and VMDK type (thick or thin). vSphere has built-in host-based replication via TCP/IP, named simply vSphere Replication. It can replicate individual VMs, and provides finer granularity than LUN-based replication. You might decide to keep the same storage vendor and protocol, but that's your choice, not something forced upon you.

We have drawn two vendors for each layer to show the message that hardware does not define the architecture. They are there to support the function of that layer (for example, Compute Function). So, you can have 10 vSphere clusters: 3 clusters could be Vendor A, and 7 clusters could be Vendor B.

We are taking the "shared-nothing architecture" approach. This is a good thing, because you contain the failure domain. Ivan Pepelnjak, an authority on data center networking architecture, states here that "Interconnected things tend to fail at the same time."

Let's summarize the key differences between SDDC and HDDC. To highlight the differences, We're assuming in this comparison the physical data center is 0% virtualized and the virtual data center is 100% virtualized. For the virtual data center, we're assuming you have also adjusted your operation, because operating a virtual data center with a physical operation mindset results in a lot of frustration and suboptimal virtualization. This means your processes and organization chart have been adapted to a virtual data center.

Disaster Recovery

As data center wide Disaster Recovery (DR) is the litmus test that defines whether your data center is HDDC or SDDC, let's start with this.

HDDC	SDDC
Data center migration is a major and expensive project.	The entire Virtual DC can be replicated and migrated. We have a customer who performed long distance vMotion over 8 weekends, hence achieving data center migration with 0 downtime.
Architecturally, DR is done on a per-application basis. Every application has its own bespoke solution.	DR is provided as a *service* by the platform. It is one solution for all applications. This enables data center-wide DR.
The standby server on the DR site is required. This increases the cost. Because the server has to be compatible with the associated production server, this increases complexity in a large environment.	No need for a standby server. The vSphere cluster on the DR site typically runs the non-production workload, which can be suspended (hibernate mode) during DR. The DR site can be of a different server brand and CPU.
DR is a manual process, relying on a run book written manually. It also requires all hands on deck. An unavailability of key IT resources when disaster strikes can impact the organization's ability to recover.	The entire DR steps can be automated. Once management decides to trigger DR, all that needs to be done is to execute the right recovery process in VMware Site Recovery Manager (SRM). No manual intervention.
A complete DR dry run is rarely done, as it is time consuming and requires production to be down.	A DR dry run can be done frequently, as it does not impact the production system. This is made by possible by having a virtual network that isolate the VMs participating in DR dry run. As a result, the dry run can even be done on the day before the actual planned DR.
The report produced after a DR exercise is manually typed. It is not possible to prove that what is documented in the Microsoft Word or Excel document is what actually happened in the data center.	The report is automatically generated, with no human intervention. It timestamps every step, and provides a status whether it was successful or not. The report can be used as audit proof

Compute Function

HDDC	SDDC
1,000 physical servers (just an example, so we can provide a comparison).	The number of VM will be more than 1,000. It may even reach 2,000 VMs . The number of VMs is higher for multiple reasons: VM sprawl; the physical server tends to run multiple applications or instances whereas VM runs only one; DR is much easier and hence, more VMs are protected.

Growth is relatively static and predictable, and normally it is just one way (adding more servers).	The number of VMs can go up and down due to dynamic provisioning.
Downtime for hardware maintenance or a technology refresh is a common job in a large environment due to component failure.	Planned downtime is eliminated with vMotion and storage vMotion.
5% to 10% average CPU utilization, especially in the CPU with a high core count.	~50% utilization for both VM and ESXi.
Racks of physical boxes, often with a top-of-rack access switch and UPS. The data center is a large consumer of power.	Rack space requirements shrink drastically as servers are consolidated and the infrastructure is converged. There is a drastic reduction in overall space and power, although power consumption per rack is higher.
Low complexity. Lots of repetitive work and coordination work, but not a lot of expertise required.	High complexity. Less quantity, but deep expertise required. A lot less number of people, but each one is an expert.
Availability and performance monitored by management tools, which normally uses an agent. It is typical for a server to have many agents.	Availability and performance monitoring happens via vCenter Server, and it's agentless for the infrastructure. All other management tools get their data from vCenter Server, not individual ESXi or VM. Application-level monitoring is typically done using agents within the Guest OS.
The word cluster generally means two or more servers joined with a heartbeat and shared storage, which is typically SAN. In another context, the word cluster means a single application using shared-nothing hardware. A typical example here is Hadoop cluster.	The word cluster has a different meaning. It's a group of ESXi hosts sharing the workload. Normally, 8 to 16 hosts, not 2 - 4.
High Availability (HA) is provided by clusterware, such as Microsoft MSCS and Veritas. Every cluster pair needs a shared storage, which is typically SAN. Typically, one service needs two physical servers with a physical network heartbeat; hence, most servers are not clustered as the cost and complexity is high.	HA is provided by vSphere HA. All VMs are protected, not just a small percentage. The need for traditional clustering software has reduced, and a new kind of clustering software emerges. It has full awareness of virtualization, and integrates with vSphere using vSphere API.
Fault Tolerance is rarely used due to cost and complexity. You need specialized hardware to achieve it.	Fault tolerance is an on-demand feature as it is software-based. For example, you can temporarily turn it on during batch jobs run.
Anti-Virus is installed on every server. Management is harder in a large environment.	Anti-Virus runs as an Agent VM per ESXi Host. It is agentless to the Guest OS and hence, is no longer visible by malware. A popular solution is Trend Micro Deep Security.

Storage Function

HDDC	SDDC
1,000 physical servers (just an example, so we can provide a comparison), where IOPS and capacity do not impact each another. A relatively static environment from a storage point of view because normally, only 10 percent of these machines are on SAN/NAS due to cost.	It can have thousands of interdependent VMs, which impact one another. A very dynamic environment where management becomes critical because almost all VMs are on a shared storage, including distributed storage.
Every server on SAN has its own dedicated LUN. Some data centers, such as databases, may have multiple LUNs.	Most VMs do not use RDM. They use VMDK and share the VMFS or NFS datastore. The VMDK files may reside in different datastores.
Storage migration is a major downtime, even within the same array. A lot of manual work is required.	Storage migration is live with storage vMotion. Intra-array is faster due to VAAI API.
Backup, especially in the x64 architecture, is done with backup agents. As SAN is relatively more expensive and SAN boot is complex at scale, backup is done via the backup LAN and with the agent installed. This creates its own problem as the backup agents have to be deployed, patched, upgraded, and managed.	The backup service is provided by the hypervisor. It is agentless as far as the VM is concerned. Most backup software use VMware VADP API to back up by taking snapshot. Windows Volume Shadow Services (VSS) provides application-consistent backups through quiesing application during backup execution. Non-VSS environments can use pre-post thaw scripts to stop necessary services prior to VM snapshot to provide crash-consistent backups of applications and underlying OS.
The backup process creates high disk I/O, impacting the application performance. Because the backup traffic is network intensive and carries sensitive data, an entire network is born for backup purposes.	Because backup is performed outside the VM, there is no performance impact on the application or Guest OS. There is also no security risk, as the Guest OS Admin cannot see the backup network.
Storage's QoS is taken care of by an array, although the array has no control over the demand of IOPS coming from servers.	Storage's QoS is taken care of by vSphere Storage I/O Control, which has full control over every VM.

Network Function

HDDC	SDDC
The access network is typically 1 GE, as it is sufficient for most servers. Typically, it is a top-of-rack entry-level switch.	The top-of-rack switch is generally replaced with the end-of-row distribution switch, as the access switch is completely virtualized. ESXi typically uses 25 GE x 2, plus an isolated ILO cable.
VLAN is normally used for segregation. This results in VLAN complexity.	VLAN is not required (traffic within the same VLAN can be controlled) for segregation by NSX.
Impacted by the spanning tree.	No Spanning Tree.
A switch must learn the MAC address as it comes with the server.	No need to learn the MAC address as it's given by vSphere.
Network QoS is provided by core switches.	Network QoS by vSphere and NSX.
DMZ Zone is physically separate. Separation is done at the IP layer. IDS/IPS deployment is normally limited in DMZ due to cost and complexity.	DMZ Zone is logically separate. Separation is not limited to IP and done at the hypervisor layer. IDS/IPS is deployed in all zones as it is also hypervisor-based.
No DR Test network is required. As a result, the same hostname cannot exist on DR Site, making a true DR Test *impossible* without shutting down production servers.	DR Test Network is required. The same hostname can exist on any site as a result. This means DR Test can be done anytime as it does not impact production.
Firewall is not part of the server. It is typically centrally located. It is not aware of the servers as it's completely independent from it.	Firewall becomes a built-in property of the VM. The firewall policy follows the VM. When a VM is vMotion-ed to another host, the policy *follows* it and is enforced by the hypervisor.
Firewall scales vertically and independently from the workload (demand from servers). This makes sizing difficult. IT ends up buying the biggest firewall they can afford, hence increasing the cost.	Firewall scales horizontally. It grows with demand, since it is deployed as part of the hypervisor (using NSX). Upfront cost is lower as there is no need to buy a pair of high-end firewall upfront.
Traffic has to be deliberately directed to the firewall. Without it, the traffic "escapes" the firewall.	All traffic passes the firewall as it's embedded into the VM and hypervisor. It cannot "escape" the firewall.
Firewall rules are typically based on the IP address. Changing the IP address equals changing the rules. This results in a database of long and complicated rules. After a while, the firewall admin dare not delete any rules as the database becomes huge and unmanageable.	Rules are not tied to the IP address or hostname. This makes rules much easier. For example, we can say that all VMs in the Contractor Desktop pool cannot talk to each other. This is just one rule. When a VM gets added to this pool, the rule is applied to it.
Load Balancer is typically centrally located. Just like the firewall, sizing becomes difficult and the cost goes higher.	Load Balancer is distributed. It scales with the demand. Adding hypervisor means adding load balancer capacity.

People & Process

How many people does it take to manage 1 rack worth of hardware?

Your answer is likely "not many." After all, it is just 1 standard rack. The entire thing barely occupies a small server room.

If your entire data center can fit inside just a few racks of equipment, that makes a small operation. It is indeed a small operation in physical world. However, in SDDC, you can achieve 3000 VM per rack from performance point of view. We are using a standard 30:1 consolidation ratio, which is possible with the latest Intel or AMD. From networking viewpoint, Ivan Pepelnjak has in fact shared back in October 2014 that "2000 VMs can easily fit onto 40 servers". He elaborates the calculation here. He further updates that in November 2015.

The above calculation takes into account your Infrastructure VM. Infrastructure functions that used to be provided by hardware (e.g. storage replication, firewall, load balancer) are now delivered as VM. You may run 100 of such VMs, depending on the type of services that your SDDC needs to provide.

HDDC	SDDC
There's a clear silo between the compute, storage, and network teams. In organizations where the IT team is big, the DR team, Windows team, and Linux team could also be separate teams. There is also a separation between the engineering, integration (projects), and operations (business as usual) teams. The team, in turn, needs layers of management. This results in rigidity in IT.	With virtualization, IT is taking the game to the next level. It's a lot more powerful than the previous architecture. When you take the game to the next level, the enemy is also stronger. In this case, the expertise required is deeper and the experience requirement is more extensive.
Relatively more headcount required in IT, with lower skills set.	Earlier, you may have needed 10 people to manage 1,000 physical servers. With virtualization, you might only need 3 people to manage 3000 VMs on 50 ESXi hosts. However, these 3 people have deeper expertise and longer experience than the 10 people combined.
DevOps is a concept that applies to developers or application team. It does not apply to Infrastructure team.	The IaaS team needs to have its own "DevOps" too. As the infrastructure becomes software, there is a need for continuous flow from Architect → Engineer → Implement → Operate → Upgrade

Epilogue

Part 4 Chapter 4

Rise of The Private Cloud Architect

I see myself as an Infrastructure Architect. After almost a decade in application world, I entered the weird and wonderful world of enterprise IT infrastructure, starting as a presale with Sun Microsystems in 2003. My job titles, roles and departments have changed many times since then, but fundamentally it's about architecting enterprise infrastructure.



This is a story of the life of a VMware Admin that I shared as an impromptu presentation at our VMUG Singapore back in 2014 and it still resonates until today.

The restaurant business provides a good analogy to your IaaS business. You, the private cloud architect, are the Chef. In that end-user environment where you work, you are the expert in producing what your customers want. You architect and design a solid platform, where your customers can confidently run their VMs. If there is an issue, you often get involved, restoring their confidence in your creation. You are seen as the VMware expert, or the virtualization expert. Yes, you may engage VMware Professional Services or Support, but they are not employees of you company. You are the employee. As far as your customers concern, the buck stops at you.

You do not sell hardware nor software; you charge your customers per VM. In fact, to ensure that your customers order the right kind of VM, you need to charge per vCPU, per vRAM and per vDisk. The chargeback model is something that I very rarely see discussed. We tend to stay in technical discussions. We need to realise we are no longer just a System Builder. We are Service Provider. By not extending our circle of influence into how Application Teams should pay for our service, we created the issue we have today (Oversized VMs, dormant VMs, VM sprawl). We need to "step out from the kitchen" from time to time. We need to be like the Chef who steps out into the dining area, building relationships with his customers, explaining the reason behind his cooking.

As the Architect, we are the best person to determine how much to charge for these. We built this environment. We know the costs, and we know the capacity. Not convinced? Put it this way, would you rather someone else determine how much your creation is worth?

We all know that IT exists because of Business. It starts with the Business. Some of the issues we have are caused by unsuitable chargeback models and incorrect Service Tiering. The VM in Tier 1 (mission critical) platform cannot cost the same as the VM in Tier 3 (non-production). I'd make sure there is distinct difference in quality between Tier 1, Tier 2 and Tier 3, so it's easy for business to choose.

Need a good example? Using the restaurant analogy, say you cook fried rice. It's your dish. You need to determine the price of the fried rice. You also need to be able to justify why you have normal fried rice and special fried rice, and why the special one costs a lot more for the same amount of food.

To me, the Chargeback model and the Service Tiering serve as Key Drivers to our Architecture. I will not consider my architecture complete unless I include these 2 in my design. We are architecting to meet the business requirements, which are "defined" in the chargeback model (e.g. the business wants a $100 VM per month, not a $100K VM per month), and service tiering (e.g. the business wants 99.999% and 3% CPU Ready).

As shared, I see a chance for us to **step up** and **step out**.

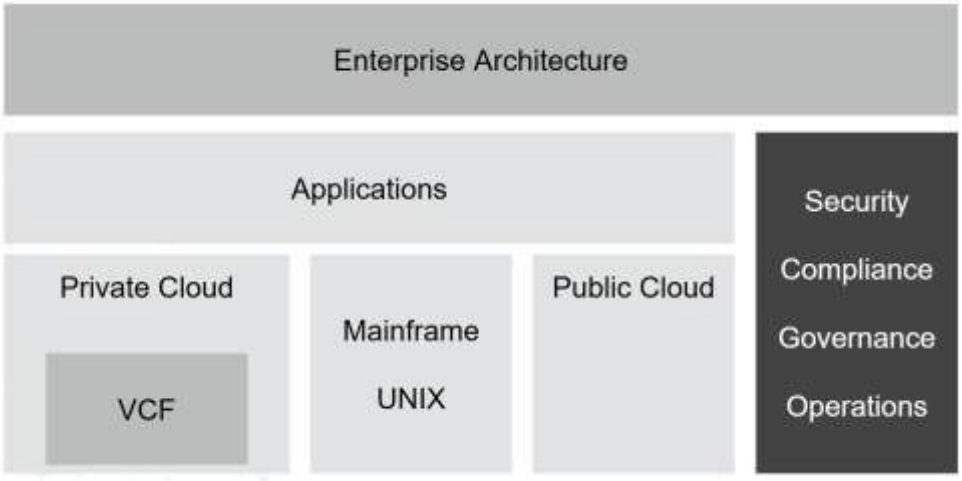
- Step out of the kitchen and network with your customers (the Application team). Educate and fix the problem at the source.
- Step up from pure IT architecture to business architecture. Architect your pricing strategy and service tiering.

The good thing about pricing is…. your benchmark is already set.

Azure, AWS, Google, and many Service Providers have already set the benchmark. Your private cloud cannot be too far from it. Too low and you will likely make a loss (it's almost impossible to beat their efficiency). Too high and you will get a complain. Another source of benchmark is to consider what it would cost to run the same applications on physical servers.

Is that all the opportunity for you?

No, you can collaborate with the Enterprise Architecture (EA) team, contributing your domain expertise. EA and Business Transformation Architect tend to look at the big picture. They are more business-centric and more strategic in their perspective.

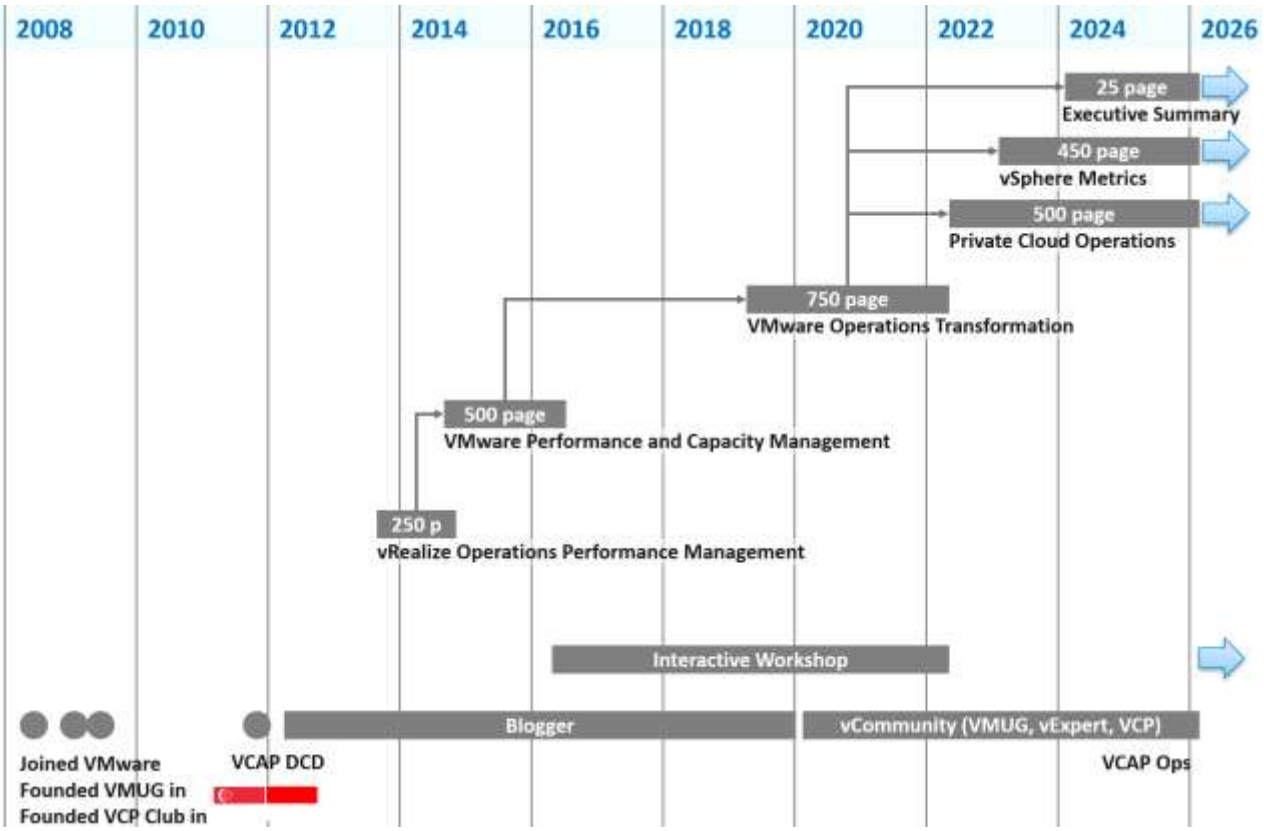


There are also departments such as security and operations. Reach out to them and collaborate.

I hope you enjoyed this perspective of your role.

Do share your life story with fellow vCommunity in various VMUG events.

History of The Books



The book you're reading has its origin on May 2008, when I joined VMware as the first Account SE for global accounts for Asia. Since all my customers had only started ESX adoption, I founded the user group in Singapore. In 2009 I founded the user group in Indonesia. As the community grew, I founded VCP Club so we could do deeper discussion.

In 2011, I was one of the first to pass the VCAP DCD exam globally as beta exam participants. In 2012 I began blogging. That knowledge from VCAP and articles from my blog set the foundation for my first book, which got published in 2014. It was ~250 pages. A good chunk of material was deferred to the second edition, hence the 2nd edition followed within 2 years. At ~500 pages, the scope was broadened and becomes less product-centric, a strategy that becomes clearer over the years.

I took a long break as the effort from writing 2 books as the 3 years of total duration were taxing, and I wanted to leapfrog the next edition. I continued blogging to gain deeper and broader knowledge. I still do a lot of researching and validating even until today, which is why the content evolves every year.

In 2019, I decided to revamp the book. It focused on overall operations management, while deepening on metrics. The 3rd edition was written in the open, meaning an update was released every month or so. After 2 years of writing, the book was finalized at ~750 pages.

In 2022, the chapters on vSphere Metrics became matured enough to be released as a separate book. My hope is we can evolve it to become VCF Metrics in the future.

The continuous release strategy worked well, and the book eventually resulted in 3 books. The goal change from operations management to operations transformation. With Broadcom's focus on private cloud, the book changed accordingly.