



# VMware® vCloud® Architecture Toolkit Cloud Bursting

Version 3.1  
January 2013

© 2013 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. This product is covered by one or more patents listed at <http://www.vmware.com/download/patents.html>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc.  
3401 Hillview Ave  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

## Contents

<b>1. Overview</b> .....	<b>5</b>
1.1 The Auto Scaling Process.....	5
1.2 Open Loop and Closed Loop Implementation Models.....	5
1.2.1 Closed Loop Systems .....	6
1.2.2 Open Loop Systems.....	7
1.2.3 Closed Loop versus Open Loop.....	8
<b>2. Sensing (Monitoring) the Service State</b> .....	<b>9</b>
2.1 Monitoring Approaches .....	9
2.1.1 Polled Monitoring.....	9
2.1.2 Stream Monitoring.....	9
2.1.3 Derived Metrics .....	10
2.1.4 Monitoring Criteria .....	10
2.1.5 End-User Monitoring .....	11
2.1.6 Infrastructure and Application Monitoring: Causal Analysis.....	11
2.1.7 Triggering the Scale Event .....	11
<b>3. Orchestration (Infrastructure Scaling)</b> .....	<b>14</b>
3.1 Scaling Localization .....	14
3.1.1 Fixed Scaling.....	15
3.1.2 Scale Everything .....	16
3.1.3 Intelligent Scaling .....	16
3.2 Scaling Orchestration.....	19
3.2.1 Foundational Requirements .....	19
3.2.2 Scaling Management.....	19
3.2.3 Adding/Removing Resources .....	21

## List of Tables

Table 1. Monitoring Criteria Categories .....	10
Table 2. Scaling Modality Benefits and Drawbacks .....	14

## List of Figures

Figure 1. Closed Loop Control System .....	6
Figure 2. Closed Loop Dynamic IaaS .....	6
Figure 3. Open Loop Control System .....	7
Figure 4. Open Loop Dynamic IaaS.....	7
Figure 5. Monitoring Process .....	9
Figure 6. Uncontrolled Scaling .....	12
Figure 7. Fixed Scaling .....	15
Figure 8. Scale Everything .....	16
Figure 9. Intelligent Scaling Flowchart .....	17
Figure 10. Intelligent Scaling.....	18
Figure 11. Scaling Resources Workflow .....	19
Figure 12. Scaling Management .....	20

## 1. Overview

*Cloud bursting* is the act of dynamically leveraging off-premise private or public compute resources in response to an increase in demand. *Auto scaling* is the act of dynamically adding local resources to a service in response to an increase in demand. Cloud bursting and auto scaling are *bursting modes* that can be triggered by an increase in demand. The resources consumed during cloud bursting or auto scaling are not explicitly dedicated to the service and are deallocated when the increase in workload normalizes.

Cloud bursting is an advanced topic that is rapidly evolving. This guide examines design guidelines, theory, and early technical insights developed to help address emerging use cases related to building an auto scaling infrastructure. The focus of this guide is specifically on the infrastructure components of auto scaling, so the document does not address the application and end-user layer implications of an auto scaling infrastructure.

### 1.1 The Auto Scaling Process

Auto scaling or cloud bursting allows the infrastructure to consume resources when they are needed and return them to the pool of available resources when they are not. This serves the end user by providing the following benefits:

- Automatic response to performance or capacity incidents.
- Reduced service delivery cost.
- Reduced outages due to human error.

The automatic or dynamic scaling of an application requires that the infrastructure provide the following components:

- Monitoring.
- Orchestration.
- A programmable API-driven infrastructure.

Each of these components can be implemented using various technologies, all providing the same function for each component. The implementation used determines how the auto scaling process is triggered and carried out, but the end result is the same. The goal of the system is to allow the application or service to autonomously remain within compliance of a service level agreement (SLA). If necessary additional resources are added to remain in compliance and meet increased demand.

### 1.2 Open Loop and Closed Loop Implementation Models

The dynamic Infrastructure as a Service (IaaS) infrastructure can be thought of in terms of two implementation models—*Open Loop* or *Closed Loop*. Regardless of the approach, a monitoring system is required to track the critical metrics used to trigger a scale out event and instruct the orchestrator to perform the scaling task.

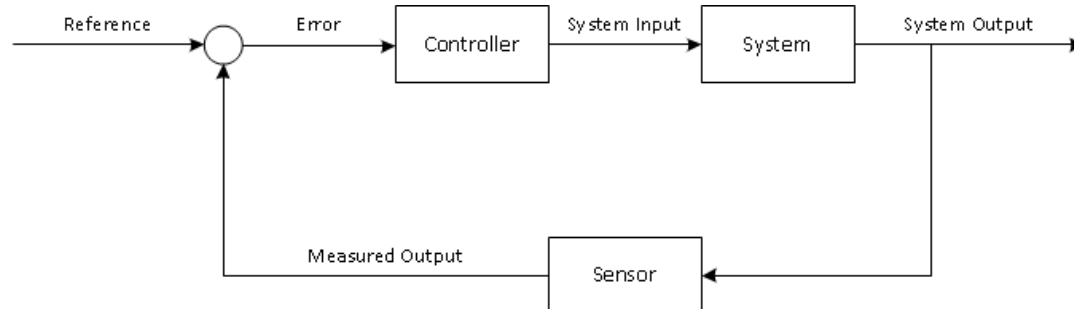
## 1.2.1 Closed Loop Systems

Closed loop control systems are those that provide feedback of the actual state of the system and compare it to the desired state of the system in order to adjust the system.

### 1.2.1.1. Control Theory

The closed loop control system is a system where the actual behavior of the system is sensed and then fed back to the controller and mixed with the *reference* or desired state of the system to adjust the system to its desired state. The objective of the control system is to calculate solutions for the proper corrective action to the system so that it can hold the set point (reference) and not oscillate around it.

**Figure 1. Closed Loop Control System**



### 1.2.1.2. Closed Loop Dynamic IaaS

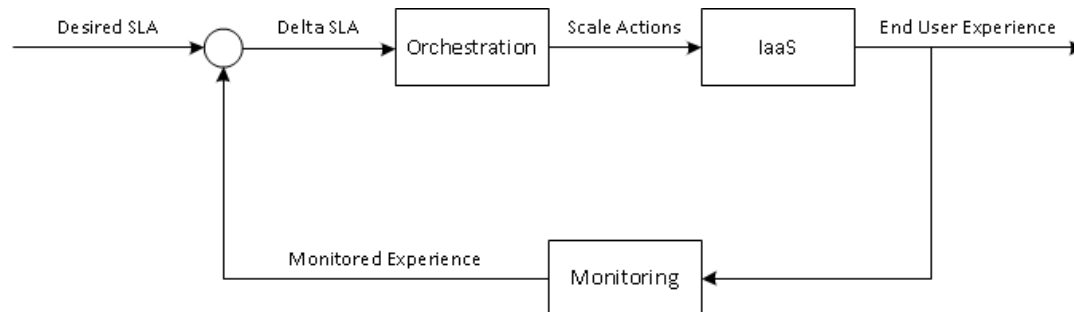
When a scale out triggering event occurs, the input parameter that triggers the event is monitored around its set point. The system increases and decreases capacity on demand to stay as close as possible to the set point for the triggering parameter.

With closed loop systems, you can evaluate the system near the set point using a PID control algorithm or similar control scheme. A simpler approach, such as *hysteresis*, can be very effective and can be implemented with less complexity and tuning.

*Hysteresis* is the dependence of a system not only on its current state but also on its past state. For example, a thermostat controlling a heater may turn the heater on when the temperature drops below  $x$  degrees, but not turn it off until the temperature rises above  $y$  degrees.

An example of a closed loop dynamic IaaS system is one in which the infrastructure is constantly monitoring the end-user experience. When an end-user experience measure drops below a desired threshold, for example, transactions taking  $> n$  milliseconds, the controller scales out the environment to compensate. The experience is checked with the new resources, and if it still is below the desired state, it continues to scale out the service. When the transaction time drops below the desired  $n$  milliseconds, the controller scales back the environment to reduce the resources consumed and continues to monitor whether the user experience is within the acceptable range.

**Figure 2. Closed Loop Dynamic IaaS**



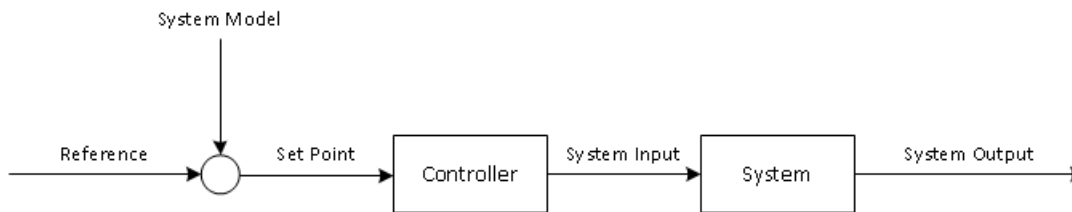
## 1.2.2 Open Loop Systems

Open loop control systems are those that do not provide feedback of the actual state of the system in order to adjust the system.

### 1.2.2.1. Control Theory

The open loop control system is a non-feedback system in which the control input to the system is determined using only the current state of the system and a model of the system. There is no feedback to determine if the system is achieving the desired output based on the reference input or set point. The system does not observe itself to correct itself and, as such, is more prone to errors and cannot compensate for disturbances to the system.

**Figure 3. Open Loop Control System**



### 1.2.2.2. Open Loop Dynamic IaaS

When a scale out triggering event occurs, the infrastructure expands its capacity through the appropriate bursting mode, either auto scaling or cloud bursting.

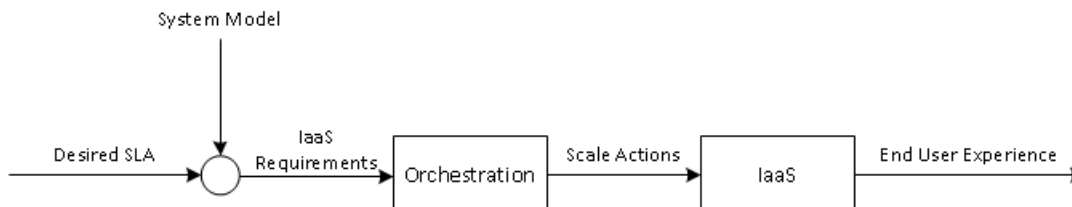
There is no feedback in the system from the usage of the new capacity to tightly control the amount of resources added or decommissioned from the service based on real world service utilization. A model of the service determines the appropriate scaling activities.

For example, a basic model of this service shows that for every 100 active sessions, one virtual machine in the web tier is required to provide a 100ms transaction time. Capacity planning data indicates that 1000 active sessions must be supported during weekdays and 250 active sessions on weekends.

On weekdays the environment scales to 10 virtual machines in the web tier (1000 sessions/100 sessions per virtual machine), and on weekends it scales to three virtual machines in the web tier (250 sessions).

The model describes the number of virtual machines per 100 sessions, but it does not account for rogue sessions that might consume significantly more resources than the typical session. It also does not account for transient spikes in resource consumption that might occur, which makes the 100 sessions per virtual machine model incorrect. In this scenario, the open loop control method does not account for the real-world state of the system, and the end-user experience degrades.

**Figure 4. Open Loop Dynamic IaaS**



### 1.2.3 Closed Loop versus Open Loop

Open loop systems have many disadvantages due to their lack of feedback from the system. With feedback in a closed loop system, you can more closely manage the state of the system relative to preferred goals, such as staying within an SLA or providing an appropriate end-user experience. Closed loop systems provide several advantages over open loop systems:

- Disturbance rejection from unforeseen increases in user load.
- Predictable performance with uncertain service models when a user does not know exactly how the service scales relative to user workload.
- Improved reference tracking where resource allocation can closely track what is needed to provide SLA compliance without over-provisioning.

Closed loop systems are recommended because of these benefits.



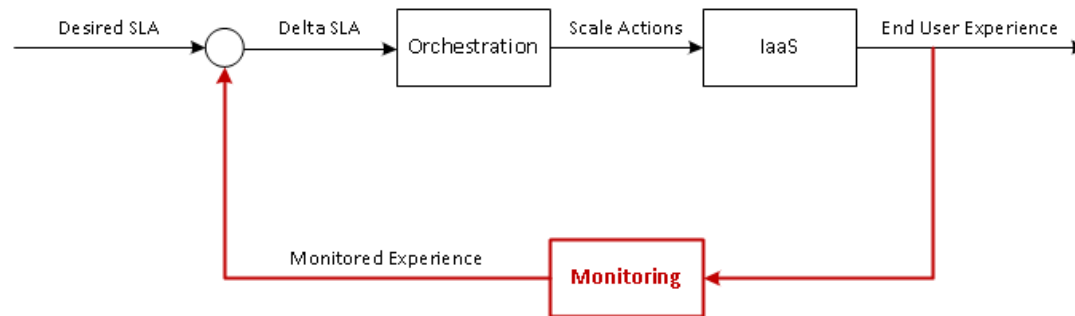
## 2. Sensing (Monitoring) the Service State

To be implemented, the control system must be able to sense (monitor) the state of the service.

### 2.1 Monitoring Approaches

*Polled monitoring* and *stream monitoring* are approaches to monitoring the service state. The following figure shows a typical monitoring process.

**Figure 5. Monitoring Process**



The observable service state is critical in implementing an effective, dynamic IaaS architecture. *Observability* is related to the possibility of observing, through measurement, the state of the service. If there is no way of understanding what the service is providing to the end users, you cannot dynamically react to that state. The fidelity of the monitoring is important because monitoring provides the information that makes it possible for the system to respond.

#### 2.1.1 Polled Monitoring

*Polled monitoring* is where the application performing the monitoring task is querying the service at a set interval and evaluating the state of the service at that moment. Polled monitoring is relatively simple to implement and typically far less costly than real-time or stream monitoring in terms of overhead and dollars. Although it is simpler and less costly than stream monitoring, polled monitoring has the following negative issues:

- Potentially long event detection periods.
- Missed events (architecture dependent).

If there is a polled monitoring solution with an interval of five minutes (300 seconds), the worst case response time to an event is 300 seconds—the worst case response time is the polling interval. This is the response time to determine that something must be done and includes the time the system takes to actually respond to the event in addition to the possible 300 seconds it took to detect the event.

#### 2.1.2 Stream Monitoring

*Stream monitoring* is passively monitoring the service by “listening” to streams of data between the application and the end user or application components. This is typically done at the network packet layer and introduces little or no overhead on the service itself. Stream monitoring provides benefits over polled monitoring. Every session is observed as it occurs, so events should not go unnoticed. However, stream monitoring is typically far more complex and costly than polled monitoring.

Though it provides the benefits of real-time visibility, stream monitoring has the following negative issues:

- Increased complexity and cost.
- It is application specific and not supported by all applications.

## 2.1.3 Derived Metrics

*Derived metrics include composite metrics and forecast metrics.*

### 2.1.3.1. Composite Metrics

Whether using polled, stream, or a combination of both monitoring techniques to observe the system, more fidelity can be provided to the results by creating macro metrics that are a function of a number of metrics to derive a composite metric that describes the system state.

### 2.1.3.2. Forecast Metrics

By using simple or complex statistical and signal analysis techniques, data can be taken from polled, streamed, or derived metrics, and it can predict what the future metrics might be. This enables the ability to provide data to the controller to make decisions ahead of the event, and to proactively take action on the system to reduce the chance of end-user impact due to slow controller response.

## 2.1.4 Monitoring Criteria

When monitoring the delivered service to understand its current behavior, the need to scale out or scale back can be done across several main categories. Each category has its own benefits and drawbacks. The ideal system considers metrics from multiple sources to make the best decisions regarding how to adapt the system to provide the preferred service level for the end user. The following table describes each category of monitoring criteria.

**Table 1. Monitoring Criteria Categories**

Category	Description	Example
Infrastructure	Utilization of specific infrastructure resources such as the following: <ul style="list-style-type: none"> <li>• CPU or memory utilization.</li> <li>• Disk latency and bandwidth.</li> <li>• Any metric that describes the health or utilization of the infrastructure.</li> </ul>	CPU utilization greater than 80% on web tier virtual machines.
Application	Consumption of application-specific resources such as active sessions.	Active sessions per web server > 200.
End User (Real)	The response time of a live user transaction exceeds acceptable levels. Measured from the perspective of real users. Can be real time.	<ul style="list-style-type: none"> <li>• Load time on a specific object &gt; 250ms</li> <li>• Page latency &gt; 100ms.</li> </ul>
End User (Synthetic)	The response time of a synthetic user transaction. Measured by executing synthetic transactions against application.	<ul style="list-style-type: none"> <li>• Load time on a specific object &gt; 250ms.</li> <li>• Complete synthetic session &gt; 5s.</li> </ul>

## 2.1.5 End-User Monitoring

Of all of the metrics that are generated by a service at all layers, *end-user experience* is a single metric that serves as an overall indicator of service health. If the end-user experience falls below a given threshold as mandated by an SLA, there is not sufficient capacity in the service to deliver the required SLA, and capacity must be added.

This approach monitors the service and uses this measure as a trigger for the scaling out and scaling back of dynamic infrastructure. Whenever the end-user experience falls below a threshold, capacity is added, and as the measure increases above the threshold, capacity is decreased. Increasing and decreasing capacity are equally important. You do not want to over-spend on infrastructure to provide a service that exceeds your SLA beyond where it provides business value based on the cost.

The drawback of using only an end-user monitoring approach is that it only indicates that there is a problem with the performance for end users. It does not provide the system any information as to where the problem is or what is causing the problem. To create a truly intelligent dynamic IaaS service, you must consider end-user experience as the key performance indicator (KPI), but infrastructure and application metrics provide the causal analysis data.

## 2.1.6 Infrastructure and Application Monitoring: Causal Analysis

When creating a dynamic infrastructure, end-user experience is almost always the most important KPI. For example, if CPU utilization on virtual machines is a constant 90%, it means that paid resources are being used efficiently. As long as the end-user experience is where it should be, high CPU or memory usage are not critical metrics. This is the target in this control model—you want to drive resource consumption on a given virtual machine as high as possible without requiring additional virtual machines, as long as end-user experience does not degrade.

When the KPI falls outside of an acceptable range, it indicates that a scaling event must be investigated. This does not explicitly mean that the incident requires scaling out. It could be an actual problem causing the KPI degradation rather than a capacity issue.

You must perform a causal analysis on the environment to determine whether to scale out, or whether the event should trigger a fault alert that requires intervention.

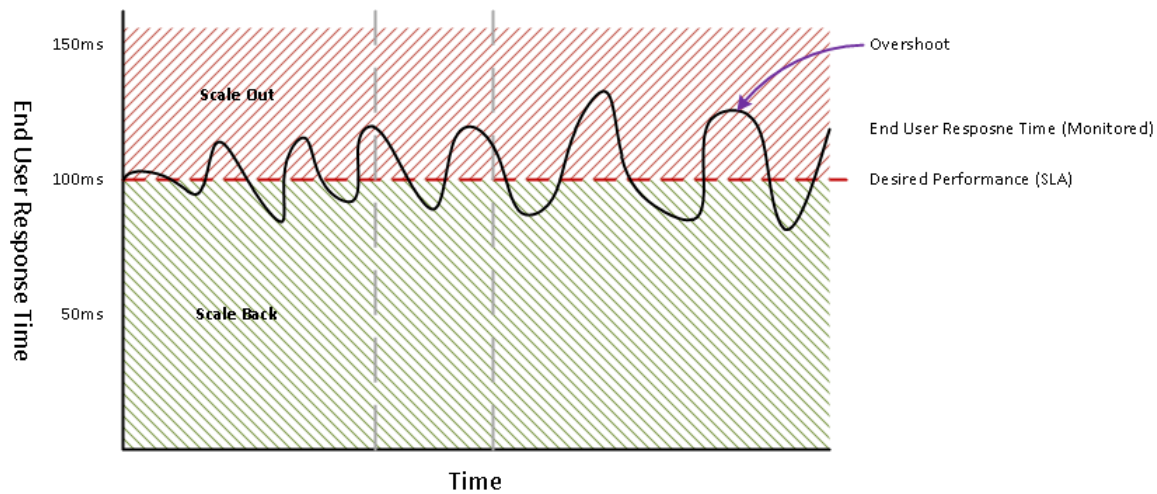
## 2.1.7 Triggering the Scale Event

Scaling can be uncontrolled, controlled, or controlled with hysteresis.

### 2.1.7.1. Uncontrolled Scaling

When triggering a scale event, you cannot determine whether to increase capacity when the threshold exceeds or falls below the set point. This causes a *ping-pong* effect, where the infrastructure is constantly scaling out and scaling back as it seeks the set point for the triggering metric. Depending on the instability of the system, this can result in significant *overshoot* while seeking the set point, where the system goes back and forth between increasingly over-provisioning resources and then decreasing them, resulting in an ever-increasing problem. The following figure illustrates uncontrolled scaling.

**Figure 6. Uncontrolled Scaling**

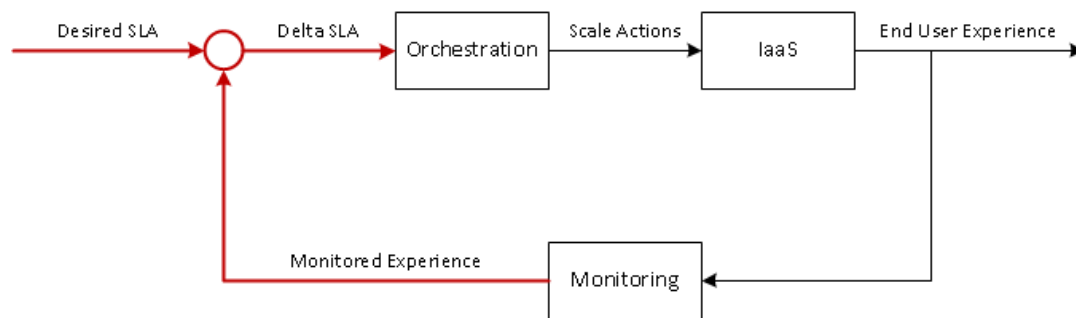


This constant expansion and contraction of resources with today's technology places an undesirable load on the infrastructure and can ultimately result in further degradation of the service.

**2.1.7.2. Controlled Scaling**

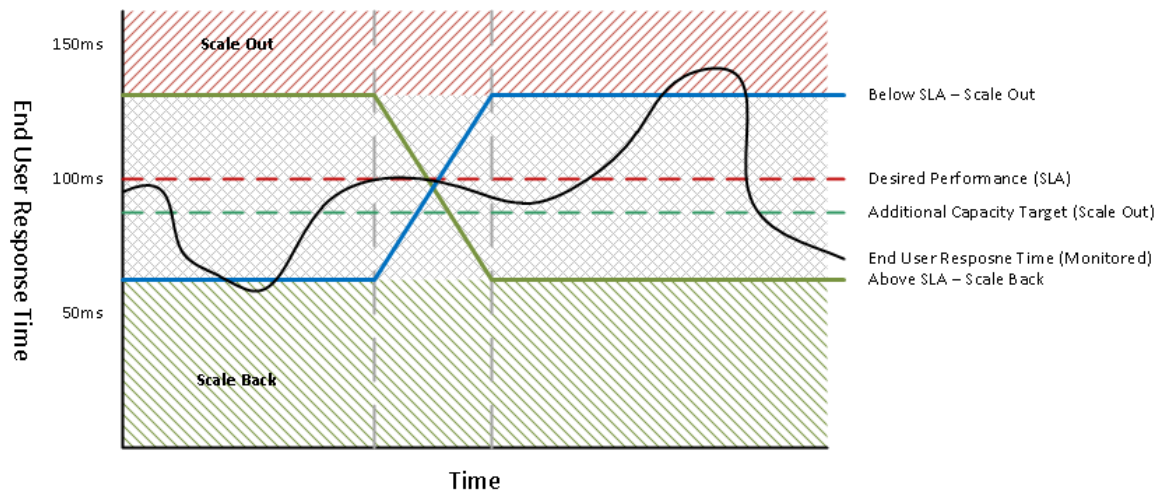
The scaling process must be controlled to provide the overall stability of the application and its underlying infrastructure.

**Figure 7. Scaling Controller**



The monitoring information, in conjunction with the preferred performance of the system (set point), must be controlled by the overall system to prevent the constant seeking of the set point. This allows the infrastructure to operate more efficiently and in a far more stable manner. The simplest control scheme for dynamic IaaS is to add hysteresis to the system.

**Figure 8. Controlled Scaling Using Hysteresis**



With this method of control, instead of aggressively seeking the set point for the end-user experience, a band can be created around it. Action is taken only when the performance falls outside of this band.

In the preceding example, as end-user experience improves in the form of decreased response time, a higher than necessary service is provided and too much capacity is consumed. This results in spending too much on the service and a scaling back of resources to get as close as possible to the SLA (set point). When response time increases and there is a reduction in service quality, when the SLA the SLA scale out threshold is reached, resources are added to bring the service level to the set point.

You can bring the service level slightly above or below the preferred set point based on your understanding of the service, how it responds to additional resources, and the cause of the increase.

By creating a dead band within the scaling model, service performance is allowed to fluctuate around the set point and not aggressively seek it, which can result in the ping-pong effect.

### 3. Orchestration (Infrastructure Scaling)

The task of scaling the infrastructure is performed by the service orchestrator. The orchestrator of the service is responsible for executing the scaling task after it is identified as necessary by the monitoring party. When scaling infrastructure, you must understand what to scale and how to scale it.

#### 3.1 Scaling Localization

Depending on the complexity and architecture of the application service, there are several approaches to scaling.

- Fixed scaling – Scale where the bottlenecks typically occur.
- Scale everything – Scale out the entire environment with each scale event.
- Intelligent scaling – Scale where the resources are needed.

**Table 2. Scaling Modality Benefits and Drawbacks**

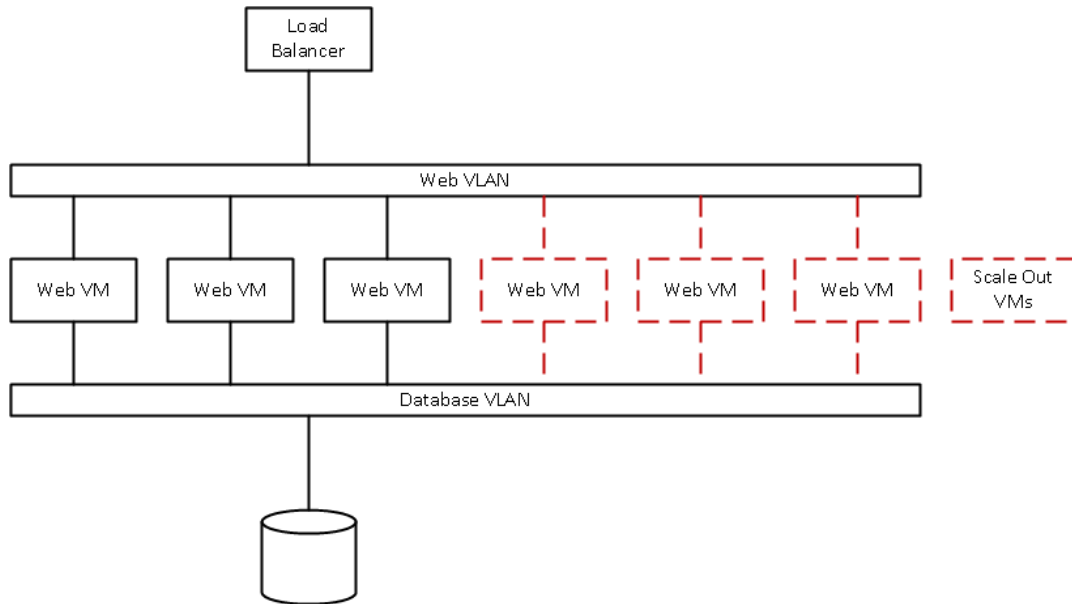
Scaling Mode	Benefits	Drawbacks
Fixed Scaling.	Simplicity.	<ul style="list-style-type: none"> <li>• Can create bottlenecks in other areas of the service.</li> <li>• Bottlenecks not within the fixed scaling components are not addressed.</li> <li>• Scaling might not address the problem, and if not managed properly, this can result in a runaway scaling event.</li> </ul>
Scale Everything.	Simplicity.	<ul style="list-style-type: none"> <li>• Can result in over-provisioning in some tiers of the service.</li> <li>• Can be far more time consuming during the orchestration phase of scaling out.</li> <li>• Can be more complicated than a fixed scaling approach due to database configuration and synchronization challenges.</li> </ul>
Intelligent Scaling.	<ul style="list-style-type: none"> <li>• Adds capacity where it is needed every time.</li> <li>• Scaling out across tiers and components dynamically avoids creating new bottlenecks.</li> <li>• Excess capacity is not added where it is not required.</li> </ul>	Complexity.

In the context of scaling, the scale remediation can scale out, scale up, or a combination of both depending on the level of complexity within the system. Make scaling design decisions based on prior knowledge of the application or services scaling characteristics.

### 3.1.1 Fixed Scaling

In the fixed scaling mode in a two-tier web application (with  $n$ -web servers and a database server), we add additional web servers to the environment as user load increases. For the scaling model, assume that the database is infinitely scalable to support the increase in web servers. Scaling the database server is a separate exercise and is addressed outside of automated scaling.

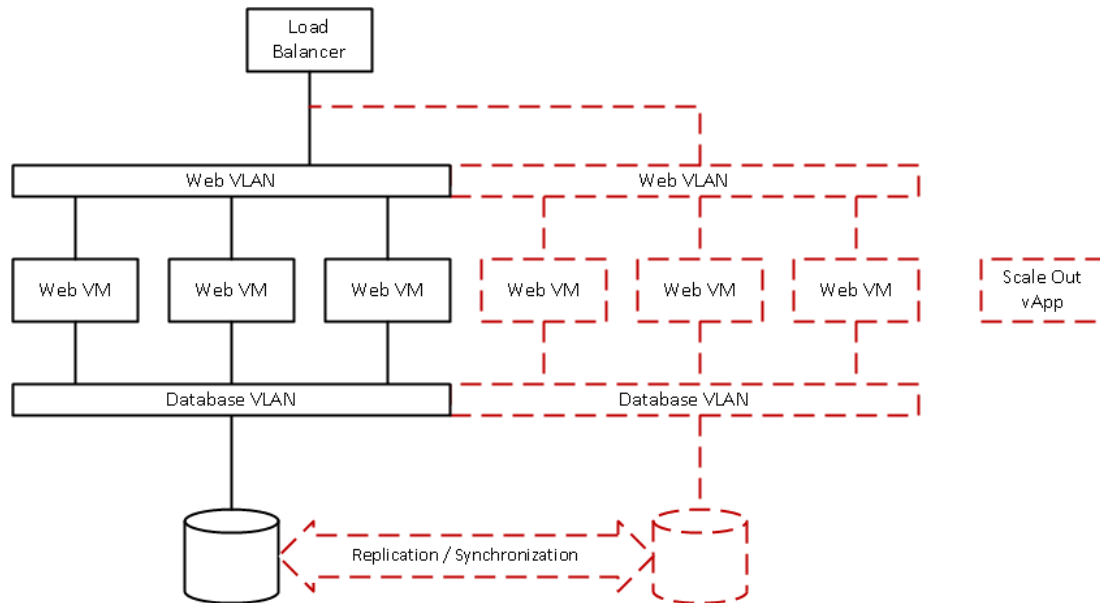
**Figure 7. Fixed Scaling**



### 3.1.2 Scale Everything

In the two-tier web application, rather than considering the database as an infinite resource, the database is more closely sized to the number of web servers within the initial deployment architecture. The entire environment is then scaled whenever a scale out event occurs. This maintains database resource alignment with the web server resources that are placing load on the database. The scale-everything model replicates the entire application service.

**Figure 8. Scale Everything**



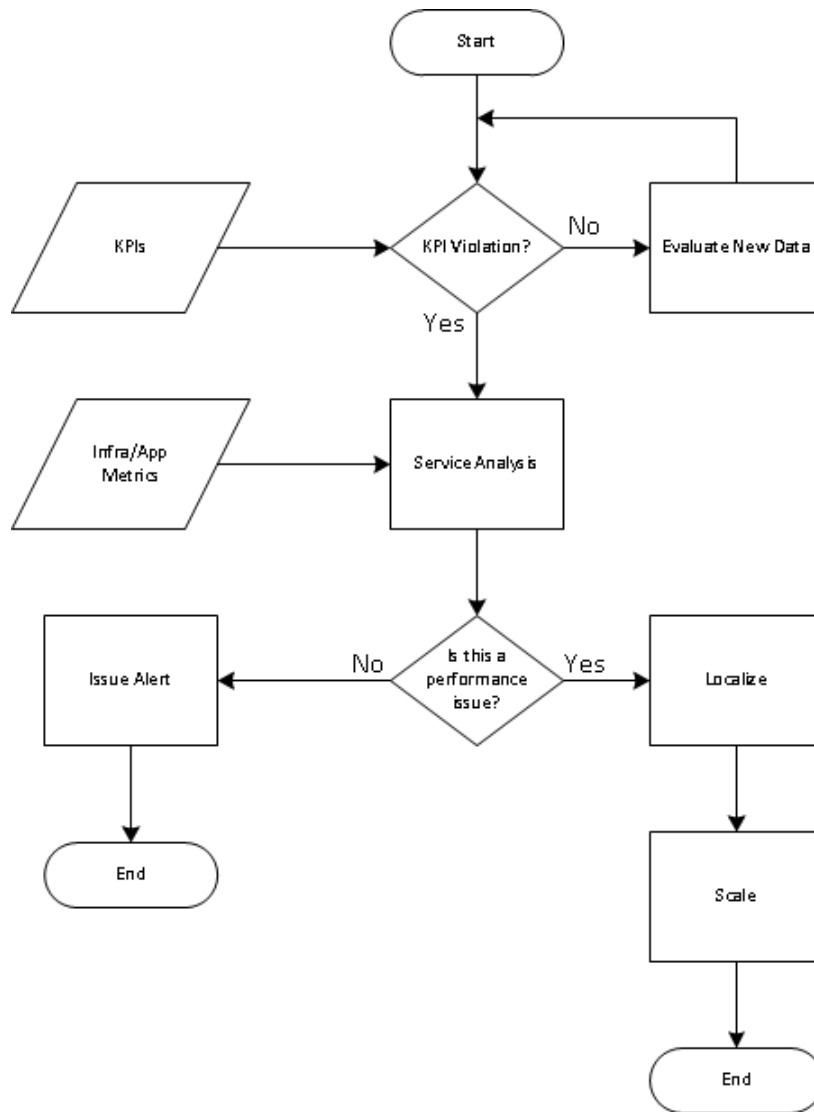
### 3.1.3 Intelligent Scaling

Intelligent scaling eliminates the drawbacks of fixed and scale everything dynamic infrastructure. This comes with the cost of increasing complexity in the system. An infrastructure that performs intelligent scaling must consider the current state of all the components within the system to identify which components are responsible for the degradation of KPI or where the service is currently over-provisioned. The system must monitor the KPI (set point) as well as the details of the system itself (infrastructure and application monitoring).

When a KPI event occurs, the system performs an analysis to determine next steps.



**Figure 9. Intelligent Scaling Flowchart**



The decision flowchart for intelligent scaling must perform the following tasks:

1. Identify a KPI violation.
2. Determine if the violation is caused by an infrastructure performance/capacity issue.
3. Localize the performance/capacity issue.
4. Issue the appropriate scaling request and scale.

### 3.1.3.1. Identify a KPI Violation

The identification of a KPI violation is common requirement of all the scale out modes, however, in all other modes, the violation triggers the scaling event. In the intelligent scaling model, the KPI violation triggers a second phase analysis or causal analysis to determine the cause of the KPI violation.

### 3.1.3.2. Determine KPI Violation Cause

To determine the KPI violation root cause and whether to scale the service, you must analyze the infrastructure and application metrics. This can be done using the following techniques:

- Trend analysis.
- Historical or baseline comparison.
- Pattern matching.

The analysis results in one of two outcomes. The violation is performance-related or capacity-related, or it is related to a problem with the service. In the case of a service problem, an alert is issued, and there should be no further action on the part of the scaling tasks.

If the analysis determines that the root cause is a bottleneck or over-provisioning, the next task is to localize the cause.

### 3.1.3.3. Localize the Cause

During the localization phase, the service metrics are further analyzed to identify the root cause of the capacity issue. Identify within which tier additional resources are needed, and which type of resource those should be.

Are the following required?

- Additional web servers?
- More database throughput?

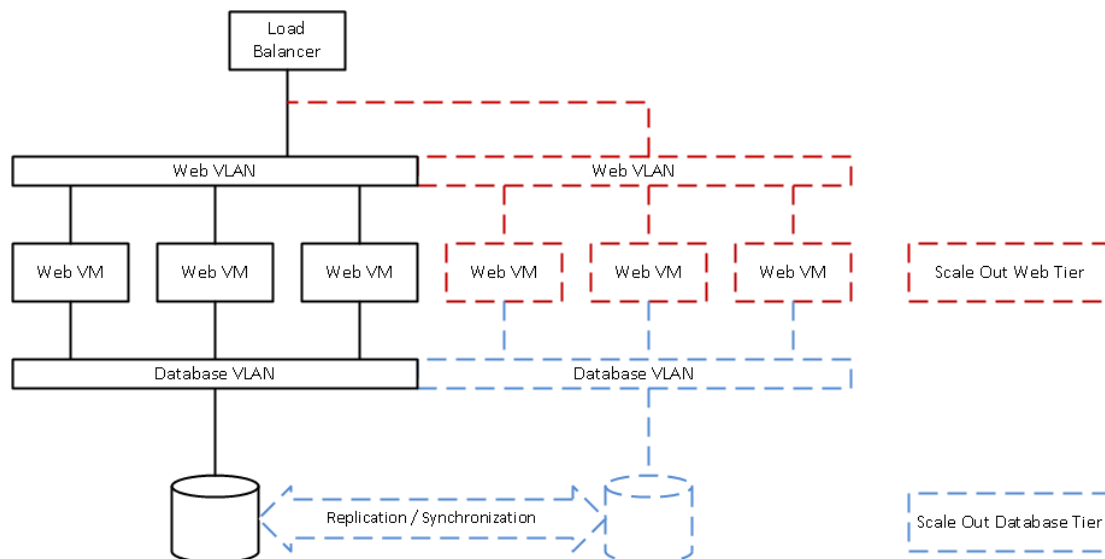
Identify the location of the capacity issue by using techniques such as the following:

- Correlation
- Anomaly detection

### 3.1.3.4. Issue the Scaling Request

After the system knows there is a performance-related or capacity-related issue, and where the issue is located, it can issue a scaling request to the orchestrator to resolve the issue. The solution might be to add additional web servers or another database node to a cluster. It might also remove capacity from the service to bring costs back in line.

**Figure 10. Intelligent Scaling**



## 3.2 Scaling Orchestration

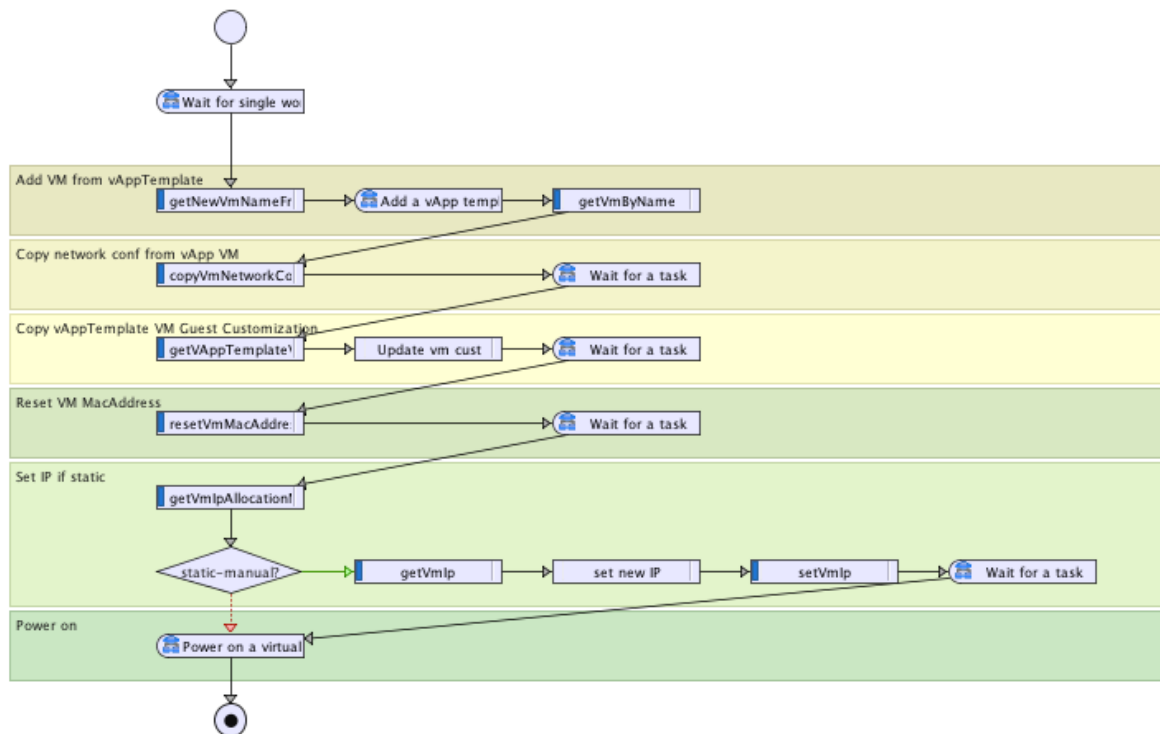
Application infrastructure scaling requires the orchestration of multiple changes to the application and underlying infrastructure. This is performed by the orchestration engine within the dynamic infrastructure.

### 3.2.1 Foundational Requirements

All scaling activities have a fundamental set of required tasks to scale any application.

- Scaling management.
- Adding/removing resources:
  - Connectivity
  - Customization
  - Starting/stopping

Figure 11. Scaling Resources Workflow



### 3.2.2 Scaling Management

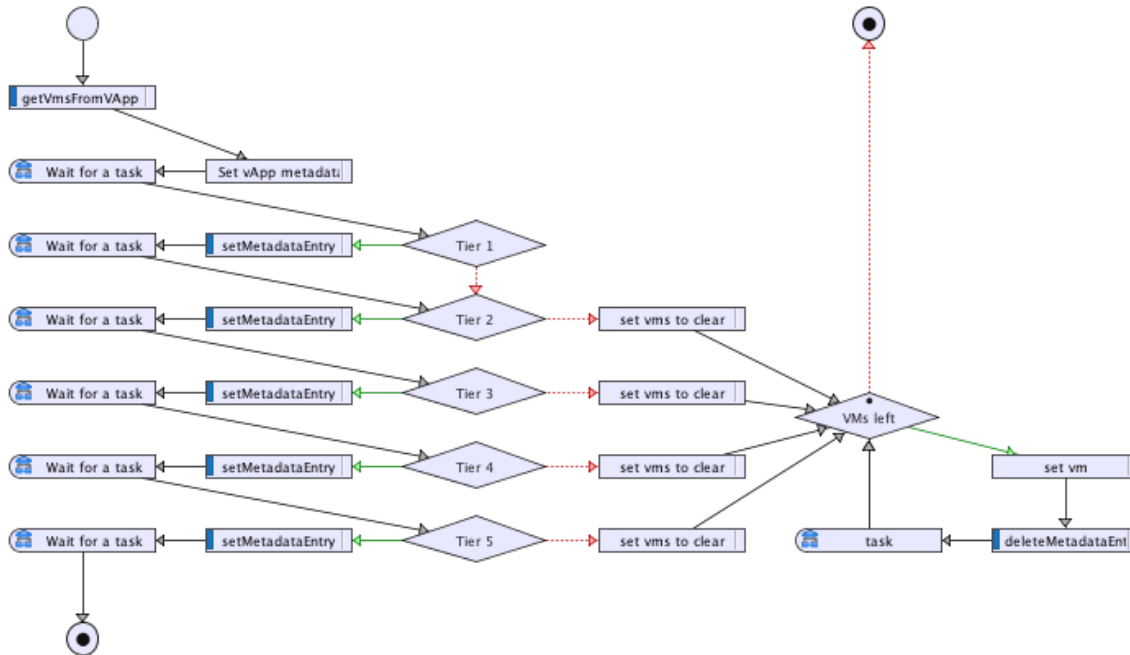
When scaling out, the orchestration of the scaling process must not only configure the resources, but must also manage the scaling task. This includes activities such as the following:

- Verifying available capacity prior to consuming additional resources.
- Coordinating or restricting parallel scaling activities on the same application.

Generally, scaling activities should be serialized to avoid issues that can arise from the parallel execution. For example, in a parallel operation, you must manage name and IP assignment across tasks to prevent duplicate names or IP addresses from being assigned to the new resources. Additionally, parallel

deployment across tiers can result in overshooting the KPI goal, resulting in the immediate triggering of a scale back activity. Parallel scaling can be implemented, but due to the complexity, it should be implemented after serial scaling has proven stable.

**Figure 12. Scaling Management**



The scaling management process is driven by directives contained within the application metadata, or by information contained within the scaling workflow. In the following example, several directives are contained as XML data within the OVF descriptor:

```
<ns2:Property ns2:value="4" ns2:userConfigurable="true"
  ns2:type="string" ns2:key="autoscaleMinTier1Instances">
  <ns2:Label/>
  <ns2:Description>Minimum number of instances for tier 1
  VMs</ns2:Description>
</ns2:Property>
```

### 3.2.3 Adding/Removing Resources

When adding resources, identify whether they are new or existing resources and what must be done with the resources.

#### 3.2.3.1. Adding Resources

- In the *fixed scaling model*, the same type of resource is always added to the service, such as additional web servers. This information can be a configuration parameter to the orchestration workflow that is set in advance by the administrator.
- In the *scale everything* model, it is the application itself that scales out, and it requires the location of the application in the service catalog from which it is derived.
- With *intelligent scaling*, the localization process identifies the resource to be scaled and passes that information to the orchestration workflow to scale that component. Depending on the application requirements, the scaling workflow might be generic enough to address all components of the application as a single workflow responsible for the scaling execution.

Due to the application complexities at each layer (web, business logic, data), a separate workflow is developed for scaling each component of the service and addressing the configuration.

#### 3.2.3.2. Connectivity

To use the newly added resources the peripheral networking components must be configured. When auto-scaling a service, the consumption of the new resources should be transparent to the user. For this reason, it is required that access pathways be automatically updated with the appropriate configuration changes so that resources can be consumed.

Configuration updates might be required for the following:

- Local load balancing.
- Global load balancing.
- Firewalls.
- Authentication.
- VPN gateways.