

**Contents**

[1 Introduction 4](#_Toc21069826)

[2 Resilient 5](#_Toc21069827)

[2.1 Limited the pod resource testing Scenario 5](#_Toc21069828)

[2.2 fault injection testing Scenario 7](#_Toc21069829)

[2.3 Test with invalid access keys 7](#_Toc21069830)

[2.4 Test with expired certificates 7](#_Toc21069831)

[2.5 Test with revoked certificates 8](#_Toc21069832)

[2.6 shut down the AIA on- premises AD/DNS 8](#_Toc21069833)

[2.7 Randomly kill pods in Kubernetes 9](#_Toc21069834)

[2.8 Testing the delay timeout configuration 9](#_Toc21069835)

[2.9 Blue-green/Canary deployment testing 10](#_Toc21069836)

[2.10 Testing Resilient patterns for BEWS 11](#_Toc21069837)

[3 Performance and Scalability 12](#_Toc21069838)

[3.1 AKS cluster and node scaling methods background information 12](#_Toc21069839)

[3.1.1 Scale an AKS cluster manually 12](#_Toc21069840)

[3.1.2 AKS Autoscaler 12](#_Toc21069841)

[3.2 Test under Cluster Node and Pod’s peak loads 13](#_Toc21069842)

[3.3 Test with Autoscaler 16](#_Toc21069843)

[3.4 Test how the sticky Session impact the Pod performance 17](#_Toc21069844)

[3.5 Real production performance testing 17](#_Toc21069845)

[4 Infrastructure connectivity 18](#_Toc21069846)

[4.1 Test network policy in AKS 18](#_Toc21069847)

[4.2 Test network connectivity for BEWS scenario 19](#_Toc21069848)

[5 Backup and restore 20](#_Toc21069849)

[5.1 Test Storage Scenario 20](#_Toc21069850)

[6 Security 21](#_Toc21069851)

[6.1 Reuse resilient test scenario 22](#_Toc21069852)

[6.2 Test invalid AD account or Invalid password scenario 22](#_Toc21069853)

[6.3 Test invalid Azure SQL database account or Invalid password scenario 22](#_Toc21069854)

[6.4 Test invalid Sybase account or Invalid password scenario 22](#_Toc21069855)

[6.5 Test Image Hardening scenario 23](#_Toc21069856)

[7 Azure Monitor 23](#_Toc21069857)

[7.1 Description about Azure Monitor 23](#_Toc21069858)

[7.2 Capabilities of Azure Monitor 25](#_Toc21069859)

[7.2.1 Monitoring VM nodes of AKS cluster 25](#_Toc21069860)

[7.2.2 Monitoring for containers 29](#_Toc21069861)

[8 Dynatrace Feature Walkthrough 32](#_Toc21069862)

[8.1 Description 32](#_Toc21069863)

[8.2 Functionality Walkthrough 33](#_Toc21069864)

[8.2.1 Azure Section 33](#_Toc21069865)

[8.2.2 Kubernetes Section 38](#_Toc21069866)

[8.3 General Problem Diagnostics 43](#_Toc21069867)

[8.3.1 Problems 43](#_Toc21069868)

[8.3.2 Smartscape topology 48](#_Toc21069869)

[8.3.3 Other monitoring perspectives 50](#_Toc21069870)

# Introduction

This document provides the general test approach and scenario for the AIA systems which run in the Azure AKS environment. We choose the BEWS system to start with as a pilot and reuse the test approach in other similar AKS base system. The following areas are covered in here:

1. Resilient
2. Performance
3. Scalability
4. Infrastructure connectivity
5. Backup and restore

# Resilient

We need to verify how the system end-to-end workload performs under intermittent failure conditions.

Measure the recovery times, and be sure that your business requirements are met. The Mean Time to Recovery (MTTR) needs to be minimized in modern day architectures.

Hence, it is beneficial to validate different failure scenarios ahead of time and to take the necessary steps to stabilize the system and make it more resilient.

Check the resiliency of the system during failures by creating following situation with fault injection:

* Delay injection
* Change access keys or expire certification
* Import expire certification
  + Check if the application shows an error for expire certification or automatically renew the certification with indication in application log
* Provide invalid access key
  + Check if the application shows an error for invalid primary access key or automatically recover by get valid backup access key
* Limit available system resources, such as RAM and CPU in POD
* A primary way to manage the compute resources within an AKS cluster is to use POD requests and limits. These requests and limits let the Kubernetes scheduler know what compute resources a pod should be assigned.
* When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources. Resource quotas are a tool for administrators to address this concern.
* Kill and Redeploy the PODs

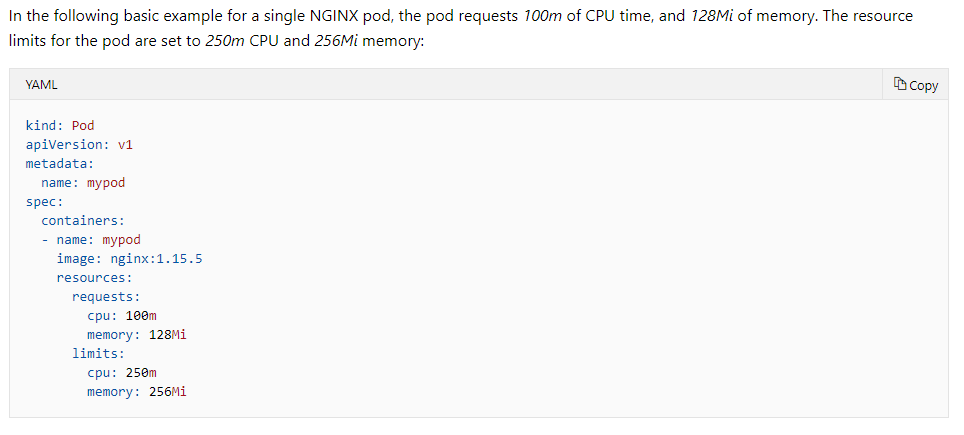
## Limited the pod resource testing Scenario

**What**

* Limit the POD resource to see if the system can handle it as expected

**How**

* Set lower POD resource requests and limits



The Kubernetes scheduler provides features that let you control the distribution of compute resources, or limit the impact of maintenance events.

The following example **YAML manifest** named *dev-app-team-quotas.yaml* sets a hard limit of a total of *10* CPUs, *20Gi* of memory, and *10* pods:

apiVersion: v1

kind: ResourceQuota

metadata:

name: dev-app-team

spec:

hard:

cpu: "10"

memory: 20Gi

pods: "10"

This resource quota in console can be applied by specifying the namespace, such as *dev-apps*:

kubectl apply -f dev-app-team-quotas.yaml --namespace dev-apps

The following resource should be considered to cover in testing:

* Compute Resource Quota
* Storage Resource Quota
* Object Count Quota

For more information about Resource Quotas:

<https://kubernetes.io/docs/concepts/policy/resource-quotas/>

**Expected**: The system should not crash, even the resource usage is very low.

**Check**: Test team need to check if the application can handle the limited resource saturation as design (such as not accept new request with error or application log indication … etc)

**When**

Run this test when there is a major deployment for each application

## fault injection testing Scenario

**What**

* Injecting an HTTP delay

**How**

* Use **Istio** to inject an HTTP delay

**When**

Run this test when there is a major deployment for each application

## Test with invalid access keys

**What**

Check how is the impact of an invalid access in Azure Key Vault (AKV)

**How**

1)Create the invalid access key and put it in AKV and verify, then we **expected**:

* The current running Pod with valid access key should keep on running
* However, the new Pod should be created and get the invalid access key, but hit error when the invalid key is used
* Also, if we kill the Pod, no new Pod can be created with certification expired error

2) Keep the old access key in AKV and then renew the API access key from the API management system.

**Expected**: The protected resource should not be accessed for current Pod and new Pod

**When**

Run this test when any code change related to using access keys for authentication

**Impact other application in same AKS**

Depending on if the access key is shared with other application

## Test with expired certificates

**What**

Check how is the impact of an expired certificates in Azure Key Vault (AKV)

**How**

Create the expired certification and put it in AKV and verify, then we **expected**:

* The current running Pod should keep on running
* However, the new Pod should be created, but hit error when the expired certification is used
* Also, if we kill the Pod, no new Pod can be created with certification expired error

**When**

Run this test when any code change related to using the expired certification for authentication

**Impact other application in same AKS**

Depending on if the expired certification is shared with other applications

## Test with revoked certificates

**What**

Check how is the impact of a revoked certificates in Azure Key Vault (AKV)

**How**

Revoke the certification in AKV, then the certification protected resource should not be accessed for current Pod and new Pod. New Pod can be started with revoked certification if the new Pod start without check the certification at the very beginning.

**When**

Run this test when any code change related to using the revoked certification for authentication

**Impact other application in same AKS**

Depending on if the revoked certification is shared with other applications

## shut down the AIA on- premises AD/DNS

**What**

Shutdown the AIA on-premises AD/DNS and verify how the application is impacted

**How**

Disconnect the network or shutdown the AD/DNS to prevent the application access the on-premises AD and verify the following condition:

* The current running Pod should keep on running
* However, the new Pod should not able to create
* Also, if we kill the Pod, no new Pod can be created

**When**

Run this test when any code change related to authentication

**Impact other application in same AKS**

Depending on if the AD/DNS is shared with other application

## Randomly kill pods in Kubernetes

**What**

A Pod is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. Verify that failures don't cascade and are handled in an isolated way, so killing a Pod should not impact other Pods.

**How**

**General:** [Kube Monkey](https://github.com/asobti/kube-monkey) is a way of testing the resilience of your system by randomly killing pods to check your system behaves properly. During functional testing automation, we can use the tool to kill the Pod randomly and expected the error are handled and retry is working well.

**For BEWS**: the user session will be killed and need to re-login, because it uses sticky session design. However, the Pods should be recreated automatically which depending on the application design. Currently, BEWS has 4 active Pods and 1 de-active Pod. We need to verify to kill the Pod randomly included the de-active Pod.

**When**

Run this test when there is a major deployment for each application

**Impact other application in same AKS**

Yes, the other application Pod may be killed too.

## Testing the delay timeout configuration

Not apply for BEWS, because BEWS has not use Istio tool yet.

**What**

Verify the timeout configuration (For example: hard-coded timeouts conflict with the timeout setting in ingress controller)

Find the hard-coded timeouts in the system that have caused the service to fail. BEWS may set the timeout by ingress controller. Test team needs to check with dev team about what is the timeout value set.

**How**

Using the following tools create delay between the request and respond:

[Istio](https://istio.io/) is an open-source service mesh that provides a key set of functionalities across the microservices in a Kubernetes cluster.

[Chaos](https://github.com/Netflix/chaosmonkey) Testing is a practice to intentionally introduce failures into the BEWS system to test the resiliency.

**Check**:

1. The delay < timeout, then system should not fail.
2. The delay = timeout, then system should not fail.
3. The delay > timeout, then system should fail with error or error handle correctly.

**When**

* + 1. Delay setting change (one off)
    2. The timeout related component code change

**Impact other application in same AKS**

The delay may impact other applications too

## Blue-green/Canary deployment testing

**What**

Your team is testing the system under real production conditions, so your team can be sure that it will function as expected when fully deployed.

**How**

Beside the SIT/UAT testing, validate the system in production using a [blue-green](https://martinfowler.com/bliki/BlueGreenDeployment.html) or [canary deployment](https://martinfowler.com/bliki/CanaryRelease.html). In the same time, we can use the monitor tool (such as Dynatrace) to collect the end-to-end system performs data, in case the intermittent failure is happened, and compare the current production.

AKS can deploy same application Pods with different container image version (such as old and new version). Only certain customers get put on new versions of the containers. They are in the “pilot” group, so when a customer logs in or goes to the system, a pilot lookup is performed. And the customer goes to the new containers only if they are in the pilot group. In the meantime, the test team can use Azure Monitor or Dynatrace to monitor the Pods with new containers and find the bugs which only happen in production environment. Once everyone in pilot is happy with how the new containers, the team do a “hard launch” and put the new containers in all Pods.

**When**

Run this test when there is a major deployment for each application

**Impact other application in same AKS**

Little impact on the other applications Pod in same node

## Testing Resilient patterns for BEWS

| **Pattern** | **What** | **How** | **When** | **Impact other App** |
| --- | --- | --- | --- | --- |
| [Bulkhead](https://docs.microsoft.com/en-us/azure/architecture/patterns/bulkhead) Isolation | Kill one of the node should not impact the system overall | Stop a node at random from a managed Azure Kubernetes Service, and no need to restart the node | The bulkhead isolation related code or setting change | Yes |
| Timeout | Verify the inbound is timeout 60 seconds for BEWS application | 1) Block the inbound access by adding deny rule in the NSG  2) remove the deny rule at 58 seconds, and expected no error  3) remove the deny rule at 60 seconds, and expected no error  4) remove the deny rule at >60 seconds, and expected timeout error in BEWS level | Timeout code or setting change | Yes, if the Timeout is set for AKS level |

# Performance and Scalability

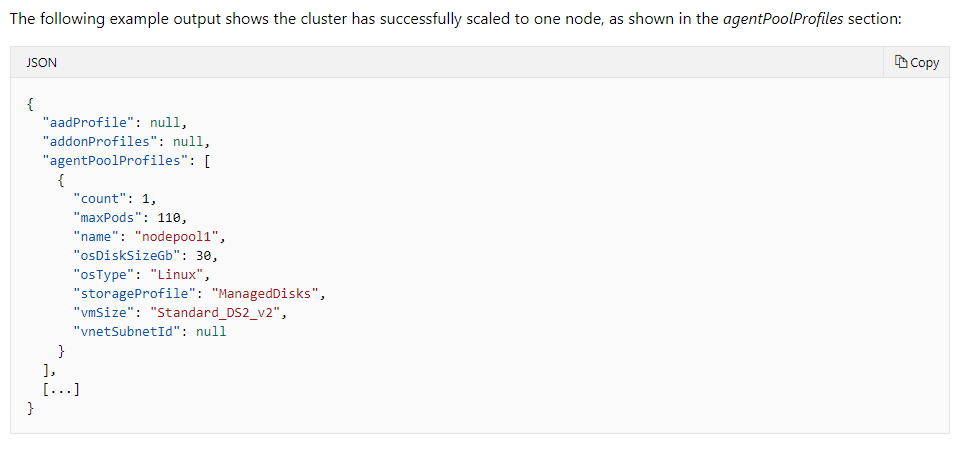
## AKS cluster and node scaling methods background information

### Scale an AKS cluster manually

We can manually scale an AKS cluster to run a different number of nodes. When you scale down, nodes are carefully [cordoned and drained](https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/) to minimize disruption to running applications. When you scale up, AKS waits until nodes are marked Ready by the Kubernetes cluster before pods are scheduled on them.

The following is the aks scale command example:

az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 1 --nodepool-name <your node pool name>

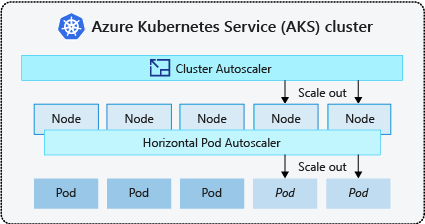


For more information, please check this website <https://docs.microsoft.com/en-us/azure/aks/scale-cluster>

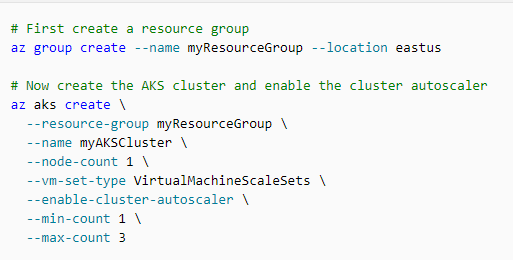
### AKS Autoscaler

AKS clusters can scale with autoscaler in one of two ways:

* The **cluster autoscaler** watches for pods that can't be scheduled on nodes because of resource constraints. The cluster automatically then increases the number of nodes.
* The **horizontal pod autoscaler** uses the Metrics Server in a Kubernetes cluster to monitor the resource demand of pods. If a service needs more resources, the number of pods is automatically increased to meet the demand.



The following example creates an AKS cluster with single node pool backed by a virtual machine scale set. It also enables the cluster autoscaler on the node pool for the cluster and sets a minimum of *1* and maximum of *3* nodes:



## Test under Cluster Node and Pod’s peak loads

**What**

BEWS has 1 cluster which contains 4 active nodes and 1 de-active node. In normal, Dev team estimates the system can have min. 12 pods in each node instance.

Currently, BEWS deployment script defined allocated resources for each pod. And Pod will only be created in different worker node, so we need to verify the load testing base line and find out the max loading for a single node in BEWS, so that we verified the worst-case situation which is only one node available. Also, we can estimation how is the loading increase base on number of nodes is added by scaling.

**How**

1. Use  [Apache JMeter](https://jmeter.apache.org/)™ and [BlazeMeter](http://www.blazemeter.com/?utm_source=blog&utm_medium=BM_blog&utm_campaign=microsoft-recommends-jmeter-and-blazemeter-as-preferred-load-testing-tools) to perform load testing for cluster.
2. The following are the different test settings:

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Number of Node** | **Number of Pod per application** | **Finding** |
| Single Node | 1 | 1 | Find the loading in worst-case situation |
| Normal | 5 (4 active node, 1 deactive node) | 1 | Base on business requirement for normal typical usage to verify the loading |
| Max | 5 (all active nodes) | 1 | Base on business requirement for peak usage to verify the loading |

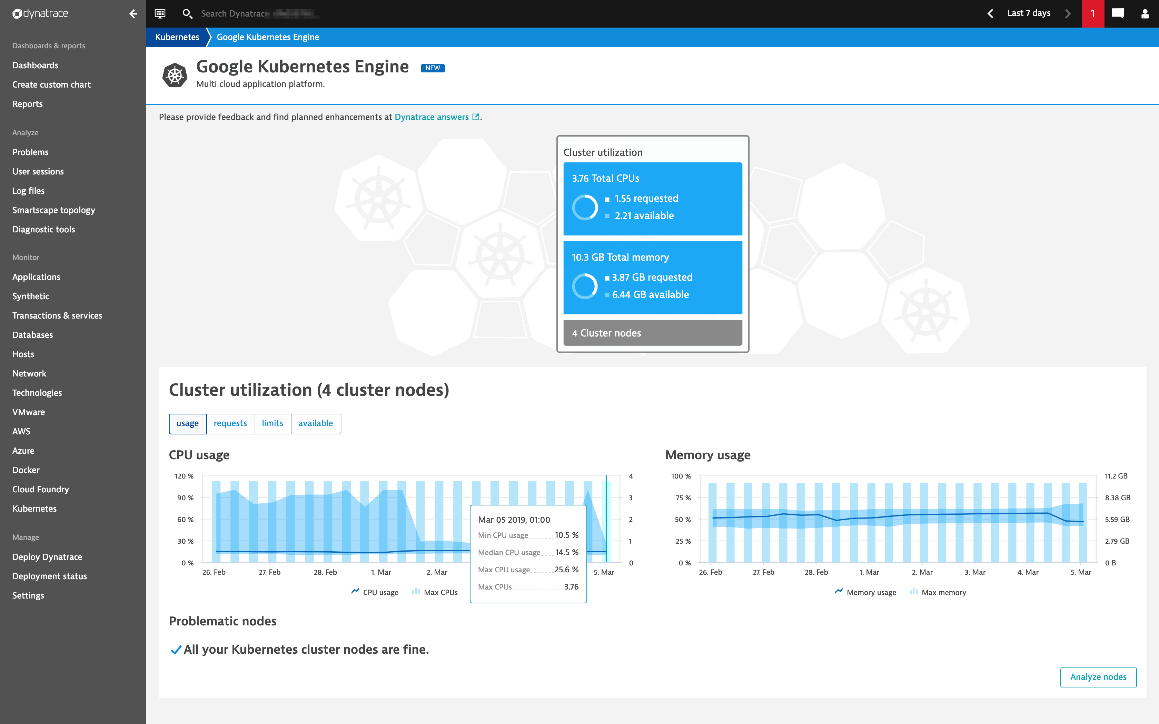
Expected the application is working as normal with good resource.

1. We can use the monitor tool (such as Dynatrace) to collect the end-to-end system performs data in different levels.

Cluster-level:

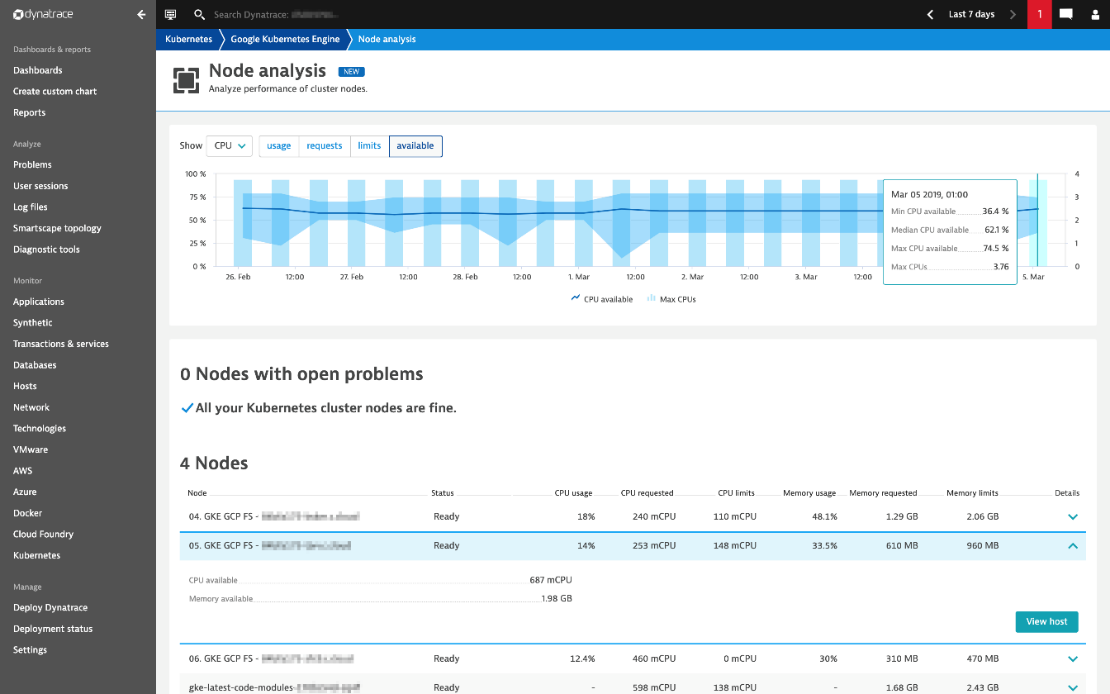
* Actual CPU/Memory usage of cluster nodes (Min, Max, Median)
* Total of CPU/Memory requests of containers running on cluster nodes (Min, Max, Median)
* Total of CPU/Memory limits of containers running on cluster nodes (Min, Max, Median)—limits may be overcommitted (i.e., over 100 %).
* Available CPU/Memory resources for running additional pods/workloads on cluster nodes (Min, Max, Median)
* Max CPUs (size of cluster in terms of CPU)
* Max memory (size of cluster in terms of memory)

In the example image below, the cluster has a total of 3.76 CPUs, where 1.55 Requests are allocated through container CPU requests and 2.21 CPUs available are still available for running further containers. The chart shows that there is at least one node with a CPU usage of 25.6% and the least utilized node has a CPU usage of 10.5%.

[](https://dt-cdn.net/wp-content/uploads/2019/03/k8s-cluster-utilization.png)

Node Level:

* how much workload can still be deployed on specific nodes:
  + CPU usage
  + CPU requested
  + CPU limits
  + Memory usage
  + Memory requested
  + Memory limits
  + CPU/Memory available metrics



**When**

Run this test when there is a major deployment for each application

**Impact other application in same AKS**

Yes, the applications share the same cluster will be impacted.

## Test with Autoscaler

**What**

If BEWS use autoscaler to scale the node and pod automatically, then we need to verify the autoscaler settings.

**How**

1. Setup single node load test environment
   1. a single node AKS environment
2. Setup auto scale test environment by **KUBE\_ENABLE\_CLUSTER\_AUTOSCALER** - it enables cluster autoscaler if set to true.
   1. Find out the loading with minimum number of nodes **KUBE\_AUTOSCALER\_MIN\_NODES** - minimum number of nodes (such as two) in the cluster.
      1. Set **KUBE\_AUTOSCALER\_MAX\_NODES** = **KUBE\_AUTOSCALER\_MIN\_NODES**
      2. Randomly kill the nodes during load testing and see if the minimum number of nodes in the cluster can be maintained in correct number
   2. Find out the loading with max number of nodes (5 nodes for BEWS application) by increase the loading until failure is occurred by [Apache JMeter](https://jmeter.apache.org/)™ and [BlazeMeter](http://www.blazemeter.com/?utm_source=blog&utm_medium=BM_blog&utm_campaign=microsoft-recommends-jmeter-and-blazemeter-as-preferred-load-testing-tools)

**KUBE\_AUTOSCALER\_MAX\_NODES** - maximum number of nodes in the cluster.

1. We can use the monitor tool (such as Azure Monitor) to collect the end-to-end system performs data in different levels.

**When**

Run this test when there is a major deployment for each application

**Impact other application in same AKS**

Yes, the applications share the same cluster will be impacted.

## Test how the sticky Session impact the Pod performance

**What**

Verify how the sticky session impact the Pod performance, and check if the same user ALWAYS go to the same Pod which may be very busy

**How**

1. Setup a load testing environment and Use  [Apache JMeter](https://jmeter.apache.org/)™ and [BlazeMeter](http://www.blazemeter.com/?utm_source=blog&utm_medium=BM_blog&utm_campaign=microsoft-recommends-jmeter-and-blazemeter-as-preferred-load-testing-tools) to perform load testing which is similar to the session 4.1 above
2. Create a single user and log on to the system, keep on using same user to perform load testing, so that the sticky session will route the request to same Pod
3. Use the monitor tool (such as Dynatrace) to collect the end-to-end system performs data in the Pod which process the request from same session

**When**

One off until the sticky session implementation is changed

**Impact other application in same AKS**

Yes, the applications share the same cluster will be impacted.

## Real production performance testing

**What**

Under the real production conditions, we measure the overall performance

**How**

Although we have production-like test environment, there are some production only issue found from time to time, because the environments are not 100% same. Verify the system in production using a [blue-green](https://martinfowler.com/bliki/BlueGreenDeployment.html) or [canary deployment](https://martinfowler.com/bliki/CanaryRelease.html) approaches, Choose the non-peak hours and run the load testing and measure the end-to-end performance and user experience by Dynatrace.

**When**

Run this test when there is a major deployment for each application

**Impact other application in same AKS**

Little impact on the other applications Pod in same node

# Infrastructure connectivity

## Test network policy in AKS

**What**

All pods in an AKS cluster can send and receive traffic without limitations, by default. To improve security, you can define rules that control the flow of traffic. Back-end applications are often only exposed to required front-end services, for example. Or, database components are only accessible to the application tiers that connect to them.

Network Policy is a Kubernetes specification that defines access policies for communication between Pods. Using Network Policies, you define an ordered set of rules to send and receive traffic and apply them to a collection of pods that match one or more label selectors.

For BEWS, no network policy is defined in AKS. And CRI (Container Runtime Interface) is not network interface / policy.

**How**

Review the Network Policy and verify if the Pods able to access the network which conflict with the Network Policy rule. Test team can use

* Kubectl command to review the policy
* wget command to verify the network access in port level.

The following is example how to set policy:

**Enable isolation**

Let’s turn on isolation in our policy-demo namespace. Calico will then prevent connections to pods in this namespace.

Running the following command creates a NetworkPolicy which implements a default deny behavior for all pods in the policy-demo namespace.



**Test Isolation**

This will prevent all access to the nginx service. We can see the effect by trying to access the service again.



For more information, please refer to

<https://docs.projectcalico.org/v3.6/security/simple-policy>

Secure traffic between pods using network policies in Azure Kubernetes Service (AKS)

<https://www.youtube.com/watch?v=131_TIa_ftI>

**When**

Run this test when there is a major deployment for each application, and the network policy change

**Impact other application in same AKS**

It may impact the other applications Pod in same node, but it should not impact the Pod in different node.

## Test network connectivity for BEWS scenario

**What**

For BEWS application (using Kubenet networking), we need to verify the following network connectivity and their inbound/outbound network policies:

* Pods to Pods network between different node
* Pods to API management gateway (Port 8556 is for API Gateway)
* Pods to other service in other vnet (such as port 8556 and SMTP service)
* Pods to on-premise connection (AD and Sybase)

**How**

After running the functional testing, use the Dynatrace’s Smartscape topology feature to get the connectivity diagram

Use the wget to verify the network connectivity. Please review the example from the session 5.2.

**When**

Run this test when there is a major deployment for each application, and the network connectivity change

**Impact other application in same AKS**

No impact

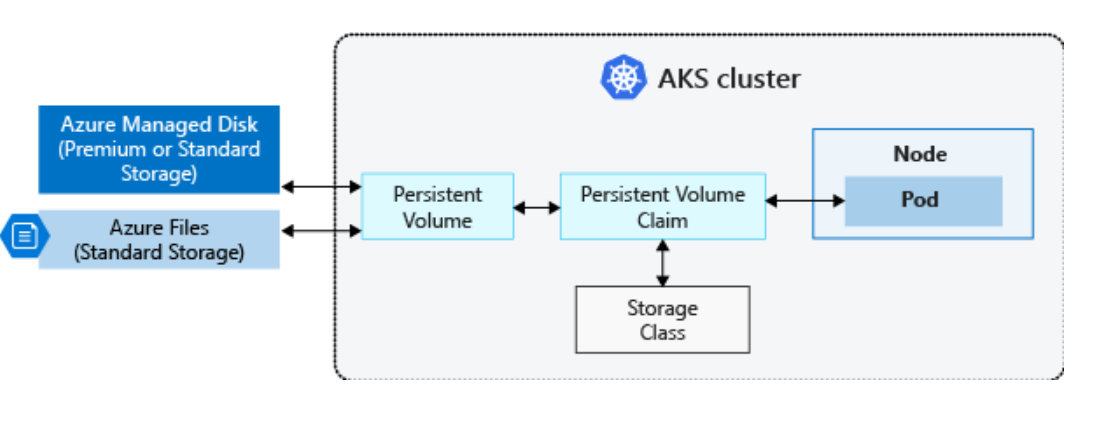
# Backup and restore

Includes choosing the appropriate storage type and node size, dynamically provisioning volumes, and data backups.

## Test Storage Scenario

**What**

When you need to attach storage to pods, you use persistent volumes. These persistent volumes can be created manually or dynamically. Manual creation of persistent volumes adds management overhead, and limits your ability to scale. Use dynamic persistent volume provisioning to simplify storage management and allow your applications to grow and scale as needed.



Persistent volume claims in an Azure Kubernetes Services (AKS) cluster

**How**

**General**: Back up your data using an appropriate tool for your storage type, such as Velero or Azure Site Recovery. Verify the integrity, and security, of those backups.  [Velero](https://github.com/heptio/velero) can back up persistent volumes along with additional cluster resources and configurations. When your applications store and consume data persisted on Azure Disks or in files, you need to take regular backups or snapshots of that data.

**For BEWS**: Because the BEWS does not use Velero and Azure Site Recovery is out of scope of this document, there is no detail verification steps.

**When**

Run this new storage implementation create

**Impact other application in same AKS**

Impact other application if the storage is share between them.

# Security

The two primary types of storage provided for volumes in AKS are backed by Azure Disks or Azure Files. To improve security, both types of storage use Azure Storage Service Encryption (SSE) by default that encrypts data at rest. Disks cannot currently be encrypted using Azure Disk Encryption at the AKS node level.

## Reuse resilient test scenario

Some of the resilient test scenarios are similar to the security scenarios. The following testing resilient test scenarios should be included in security testing as well:

3.2 Test with invalid access keys

3.3 Test with expired certificates

3.4 Test with revoked certificates

## Test invalid AD account or Invalid password scenario

**What**

Make sure only the AD account with right permission can access the resource

**How**

Create the invalid AD account which is used by the node, verify if the invalid AD account can access the resource. Expected an error will be handled, and log to the application log

**When**

The security related code change or the AD server changed

**Impact other application in same AKS**

No impact except the user account was shared with other application

## Test invalid Azure SQL database account or Invalid password scenario

**What**

Make sure only the Azure SQL database account with right permission can access the resource in database

**How**

Create the invalid Azure SQL database account which is used by the node, verify if the invalid Azure SQL database account can access the database resource. Expected an error will be handled, and log to the application log

**When**

The security related code change or the Azure SQL database server changed

**Impact other application in same AKS**

No impact except the Azure SQL database account was shared with other application

## Test invalid Sybase account or Invalid password scenario

**What**

Make sure only the Sybase database account with right permission can access the resource in Sybase database

**How**

Create the invalid Sybase SQL database account which is used by the node, verify if the invalid Sybase SQL database account can access the database resource. Expected an error will be handled, and log to the application log

**When**

The security related code change or the Sybase SQL database server changed

**Impact other application in same AKS**

No impact

## Test Image Hardening scenario

AIA Group only has hardening guideline for Red Hat Enterprise Linux, SUSE Enterprise Linux, and Microsoft Windows Server. These are licensed OS which is not suitable for Container, so the BEWS’s node image is not hardening.

**What**

Make sure the image is hardening as design

**How**

Check the hardening setting one by one

**When**

After the image is created

**Impact other application in same AKS**

No impact

# Azure Monitor

## Description about Azure Monitor

Azure Monitor is the all-in-one Cloud native monitoring solution which is provided by Microsoft Azure. Aim to maximize the availability and performance of enterprise application, it delivers comprehensive solution for collecting, analyzing and act as a centralize telemetry to provide visibility of the cloud environment.

From high-level view, Azure monitor will handle two fundamental type of data stores, metrics and log. Collected data will be handled to provide below functions:

* Insights
* Visualize
* Analyze
* Response
* Integrate

This document will first introduce few main features of Azure Monitor and then provide some general use cases for daily operations. By facilitating IT operation tasks with Azure Monitor, end users can easily take advantage of additional visualizations.

## Capabilities of Azure Monitor

### Monitoring VM nodes of AKS cluster

Azure Monitor for VMs monitors Azure virtual machines (VM) and virtual machine scale sets at scale. It analyzes the performance and health of AKS’s nodes and monitors their processes and dependencies on other resources and external processes.

**Key feature of Azure Monitor for VMs:**

* **Health** - Logical components of Azure VMs that run Windows and Linux: Are measured against pre-configured health criteria, and alert will be sent out when configurable threshold reached.
* **Performance** - Pre-defined trending performance charts: Display core performance metrics from the guest VM operating system.
* **Map -** Dependency map: Displays the interconnected components with the VM from various resource groups and subscriptions.

**Prerequisites**

* Require enabling this feature on VM nodes
* Need a Log Analytics workspace to store those data
* Health/ Map agent will be installed on the VM as extension when onboard this feature

**Data Usage**

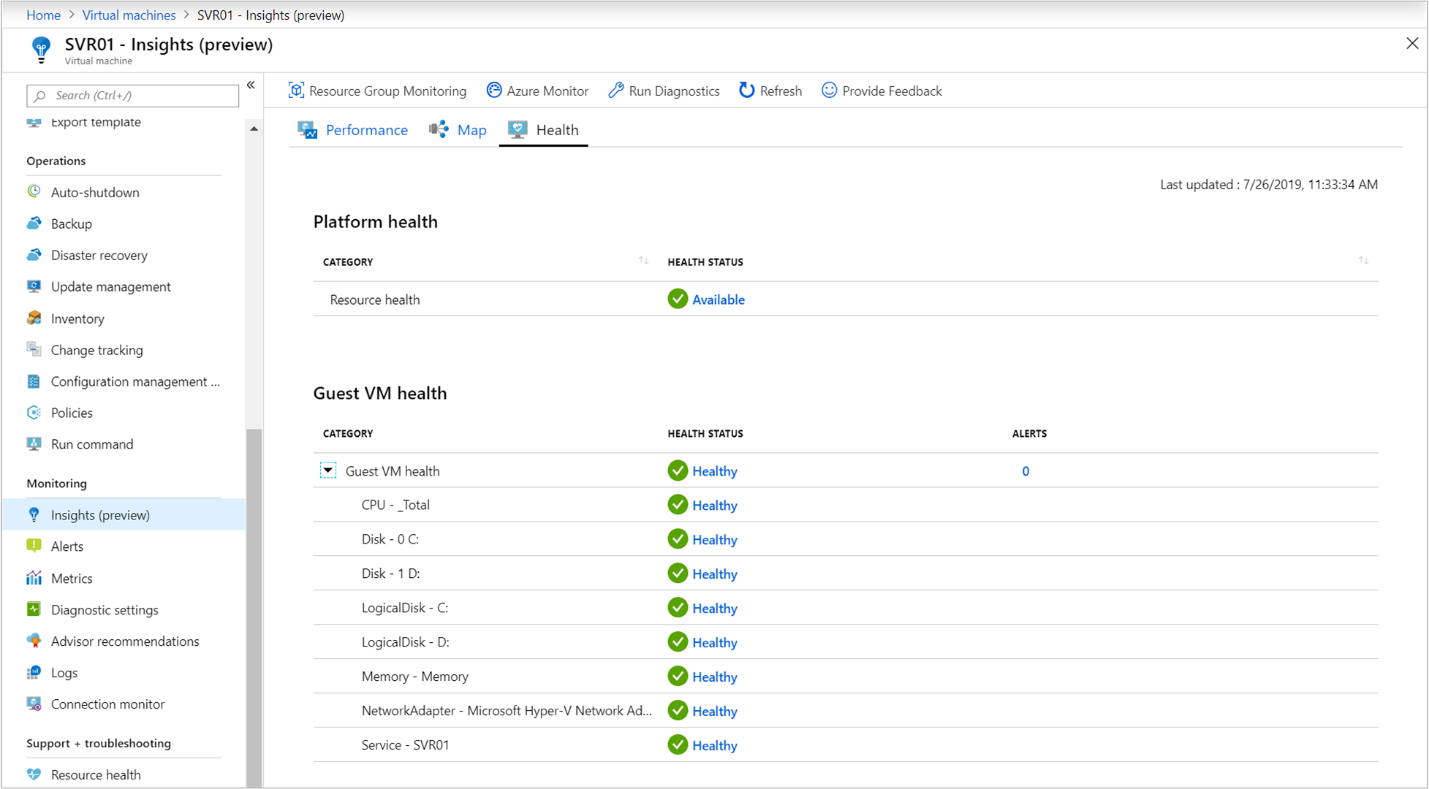
After the Azure Monitor on VMs has been enabled, the data that's collected by the VMs is ingested and stored in Azure Monitor. Health criteria metrics are stored in Azure Monitor in a time-series database, performance and dependency data collected are stored in a Log Analytics workspace.

**Onboard Azure Monitoring on Azure VM**

Refer to below section to enable this feature.

3.1 Enable Azure monitor on AKS nodes

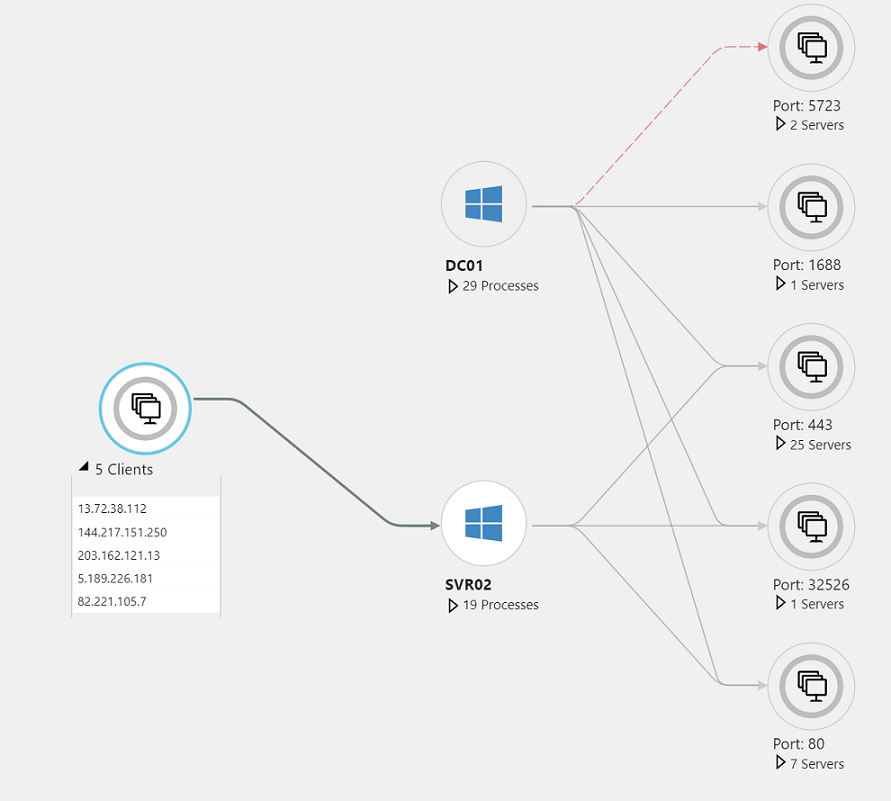
**Screencap example on Health on AKS nodes**

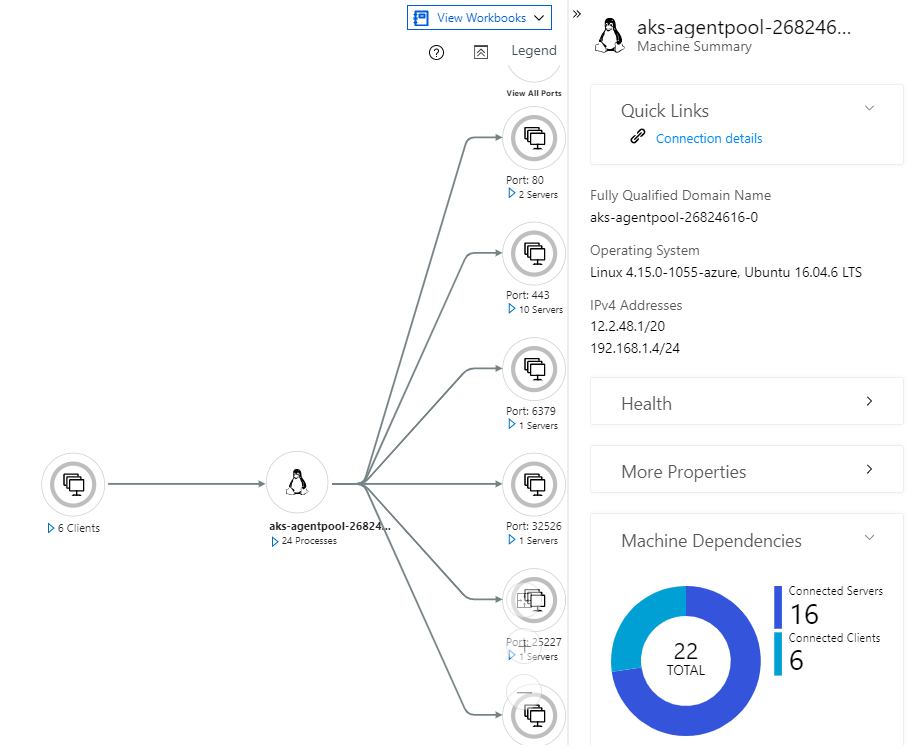


**Screencap example of show “MAP” with Azure Monitor**

The Map feature visualizes the VM dependencies by discovering running processes that have:

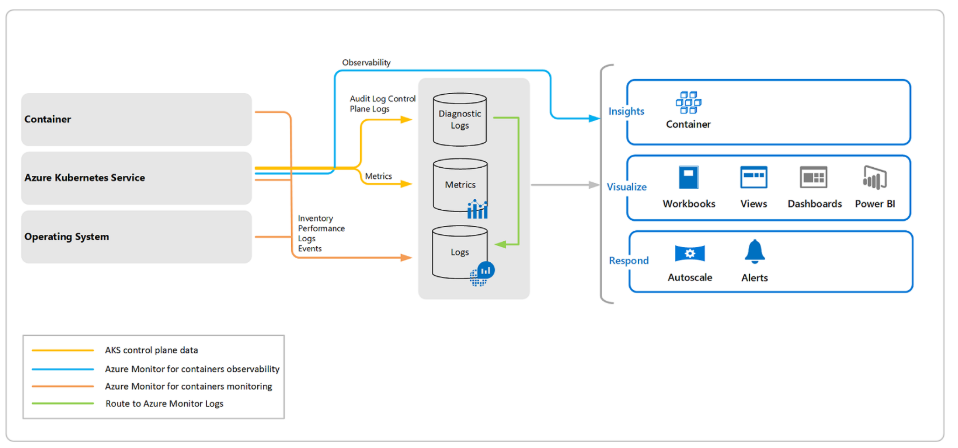
* Active network connections between servers.
* Inbound and outbound connection latency.
* Ports across any TCP-connected architecture over a specified time range.





### Monitoring for containers

Azure Monitor for containers gives performance visibility by collecting memory and processor metrics from **controllers, nodes, and containers** that are available in Kubernetes through the Metrics API. Container logs are also collected. After enabling the monitoring from Kubernetes clusters, metrics and logs are automatically collected through a containerized version of the Log Analytics agent for Linux. Metrics are written to the metrics store and log data is written to the logs store associated with your Log Analytics workspace.



**Key features**

With Azure Monitor for containers you can:

* Identify AKS containers that are running on the node and their average processor and memory utilization. This knowledge can help you identify resource bottlenecks.
* Identify where the container resides in a controller or a pod. This knowledge can help you view the controller's or pod's overall performance.
* Review the resource utilization of workloads running on the host that are unrelated to the standard processes that support the pod.
* Understand the behavior of the cluster under average and heaviest loads. This knowledge can help you identify capacity needs and determine the maximum load that the cluster can sustain.
* Configure alerts to proactively notify you or record it when CPU and memory utilization on nodes or containers exceed your thresholds.

**Prerequisites**

* Require enabling this feature on the AKS cluster
* Need a Log Analytics workspace to store those data

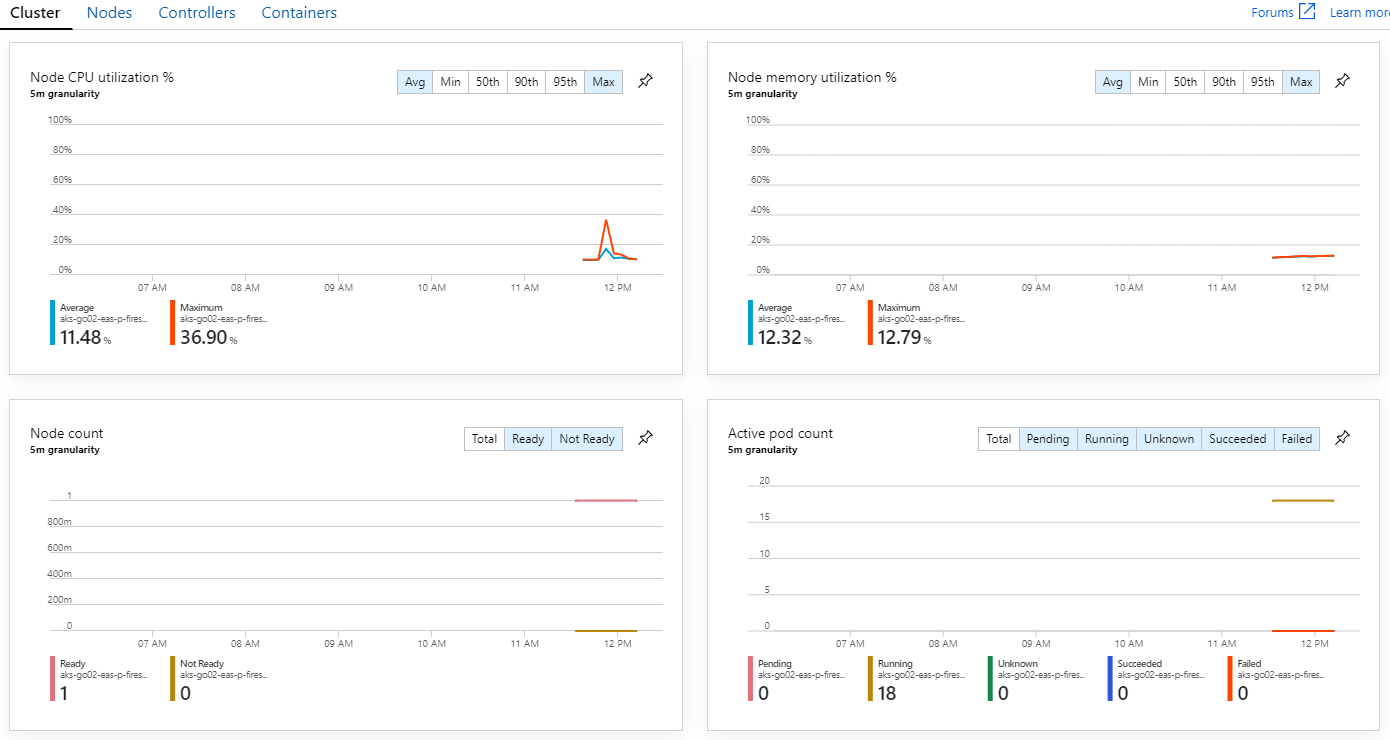
**Onboard Azure Monitoring on Azure VM**

Refer to below section to enable this feature.

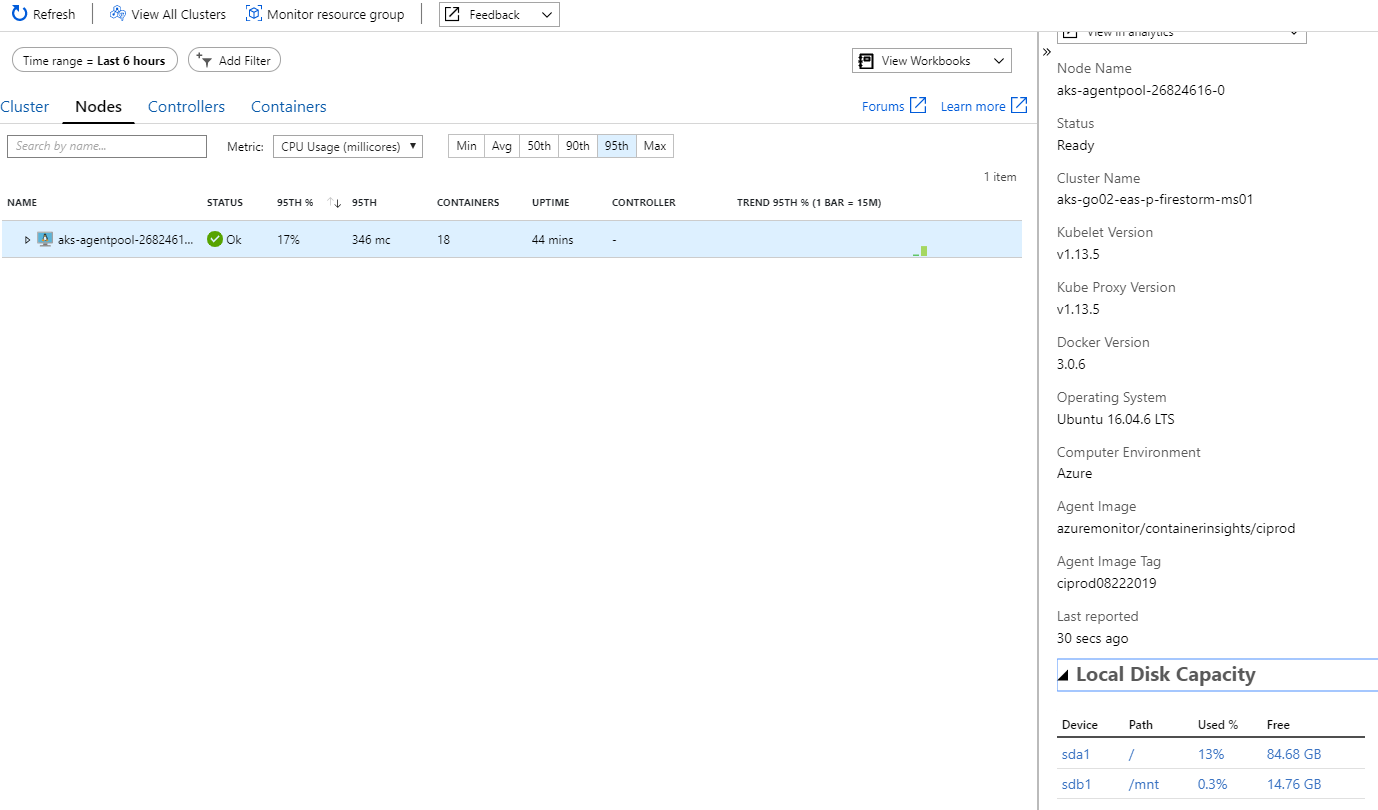
3.2 Onboarding Azure Monitor on AKS

**Screencap example of AKS container monitoring**

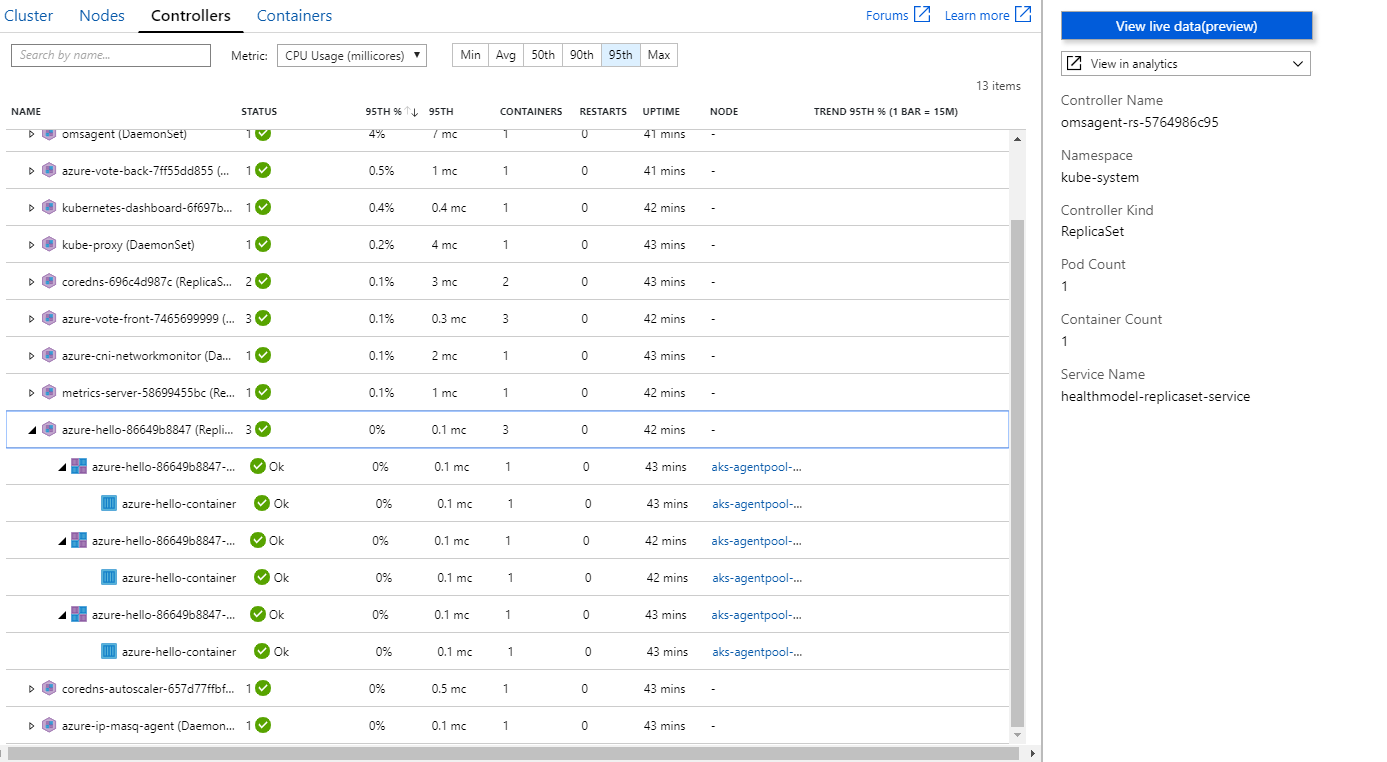
Utilization on the clusters



Utilization on the nodes



Utilization info of controllers



# Dynatrace Feature Walkthrough

## Description

Dynatrace, as a centralized AI-powered monitoring platform, provides a single point of view to facilitate users on monitoring and troubleshooting their Infrastructure and Application problems from different dimensions: Application, Transaction, Database, Host, Network, User Session, etc.

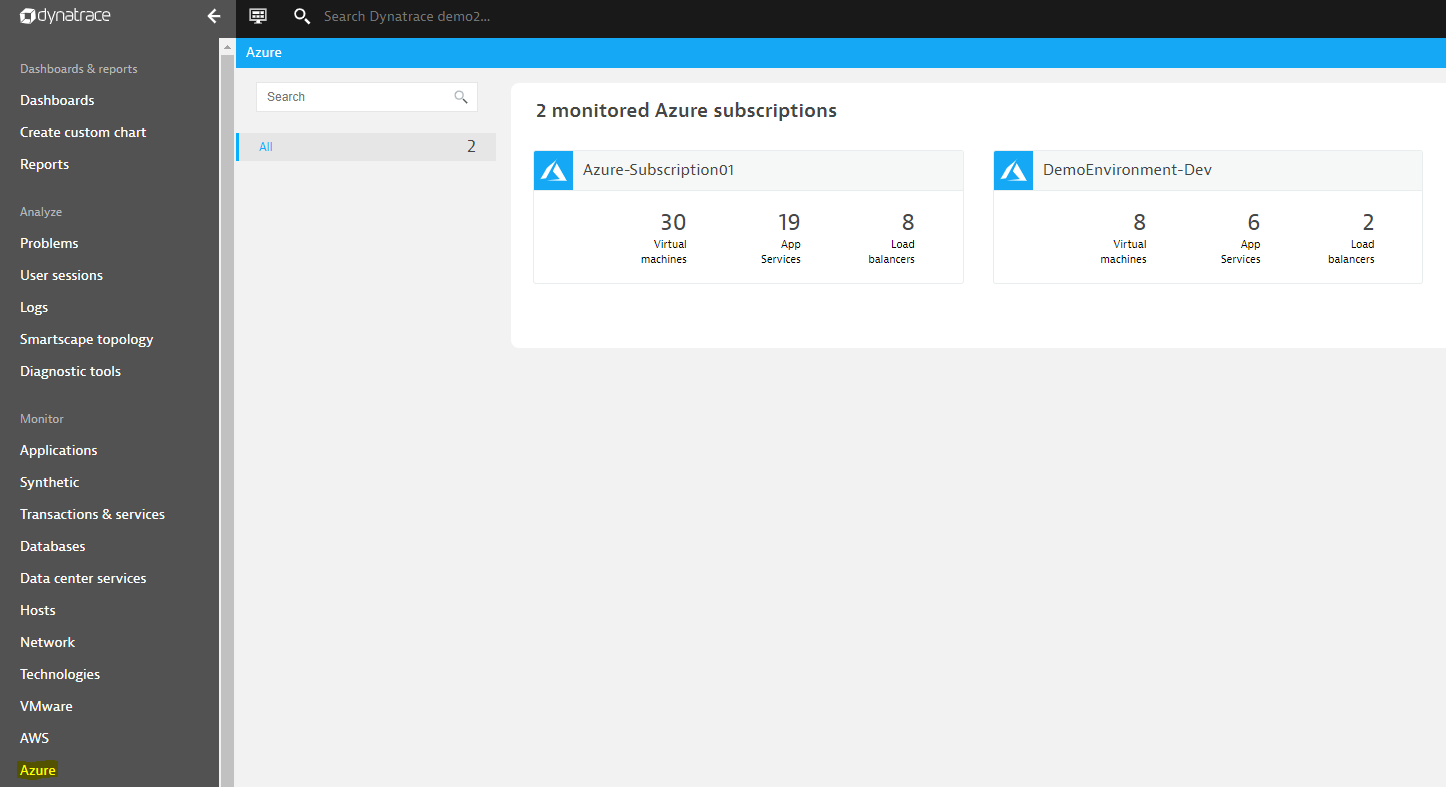
This session will be divided into two parts, the first part will focus on exploring the functionality on Azure and Kubernetes monitoring features natively supported by Dynatrace. The second part will introduce a general problem diagnostic practice that utilized different Dynatrace components, providing a full-dimension visibility to users on understanding the issue happened in their own environment, and facilitate the decision making on following up the issue.

## Functionality Walkthrough

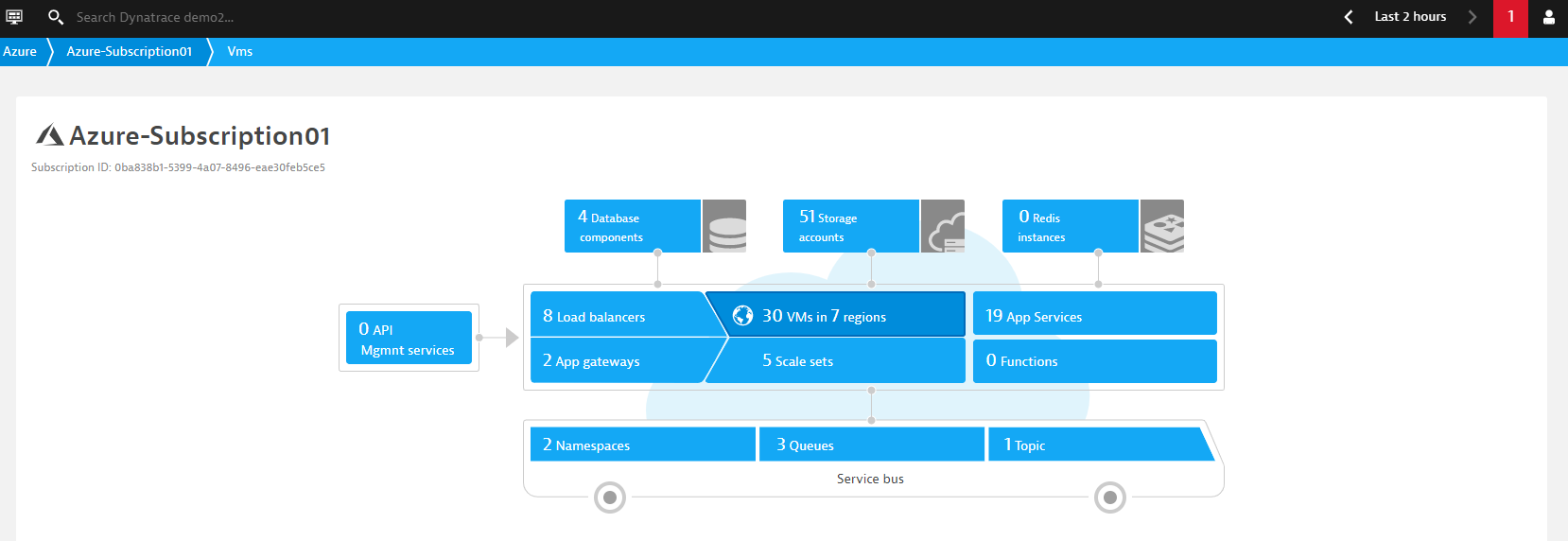
This section will first cover the monitoring capability on Azure components, followed by that on Kubernetes resources.

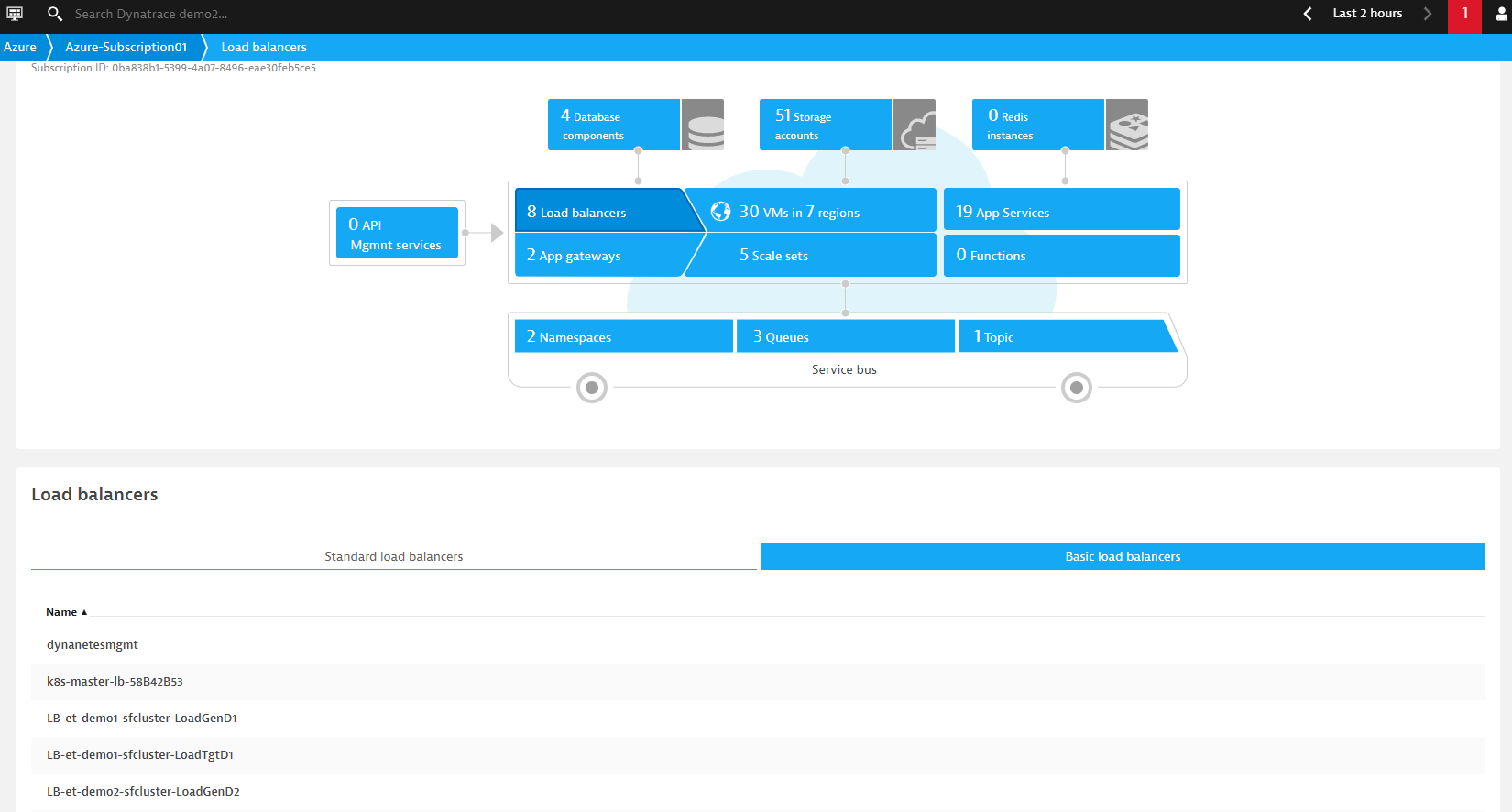
### Azure Section

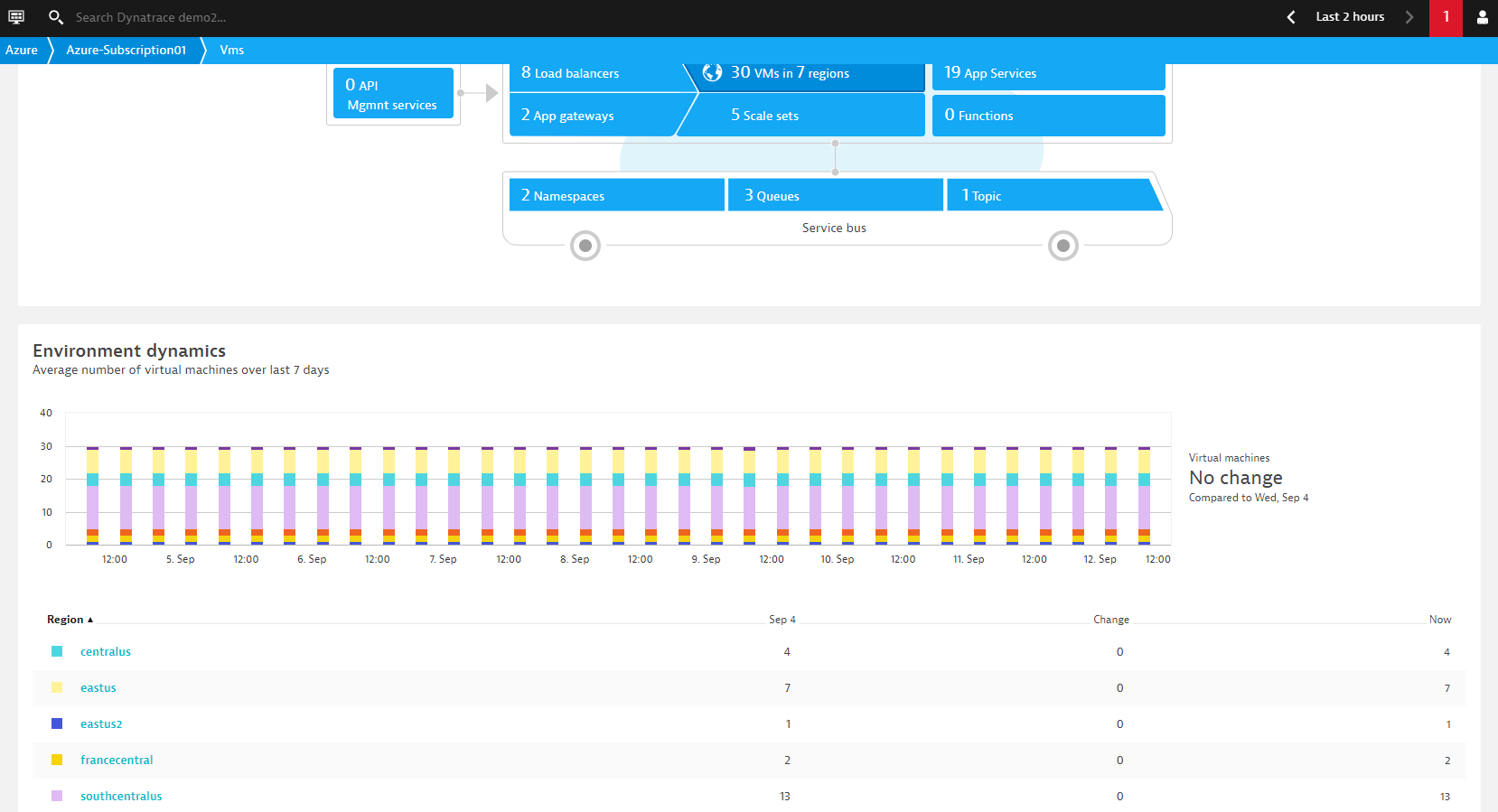
1. Azure monitoring feature is natively supported by Dynatrace. Select Azure in the left menu bar to start the investigation. The monitored Azure subscriptions are listed in a summary page, click into the subscription for more details.

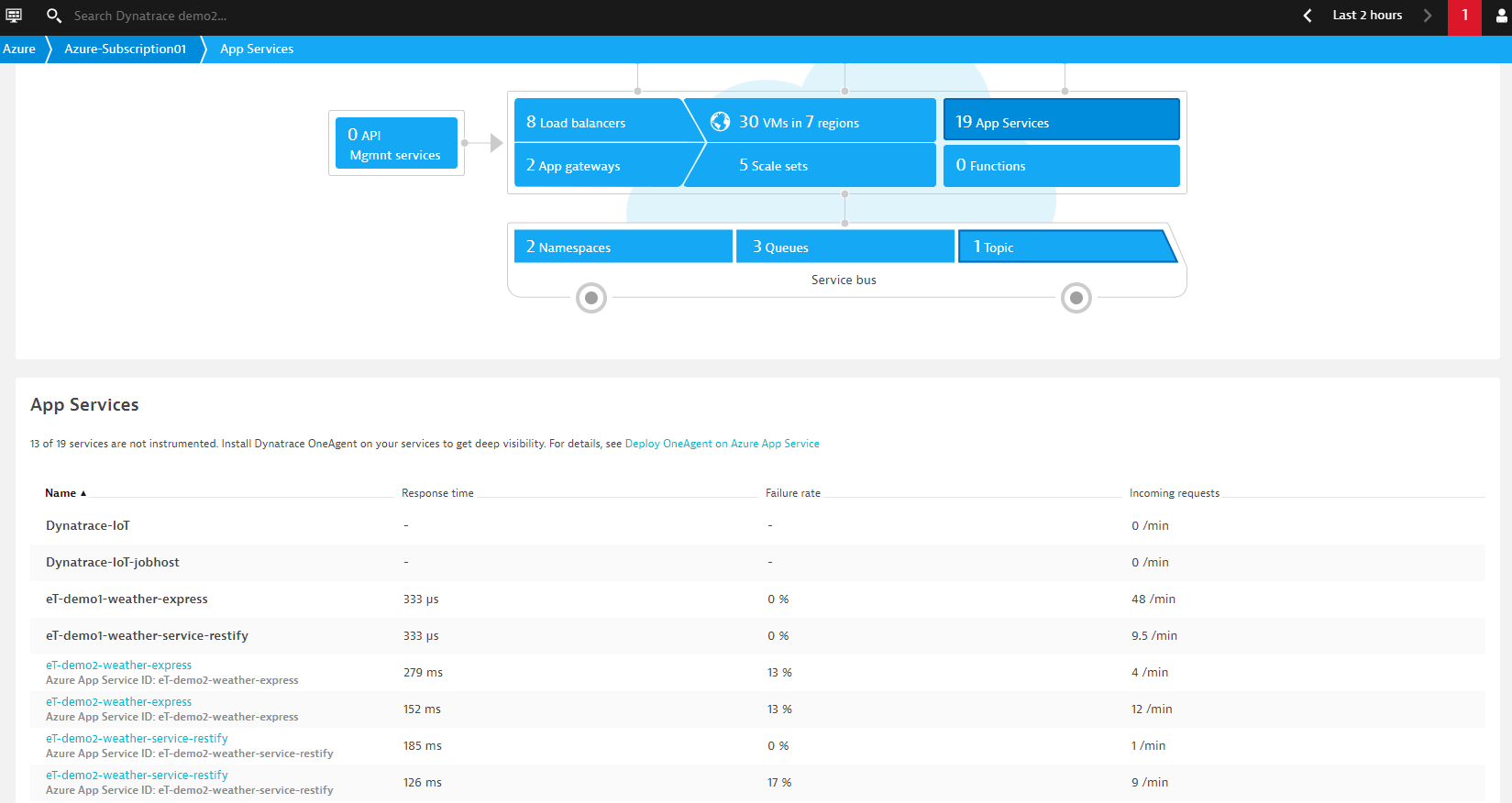


2. The component overview will be displayed in this section, click into each component to obtain the details.

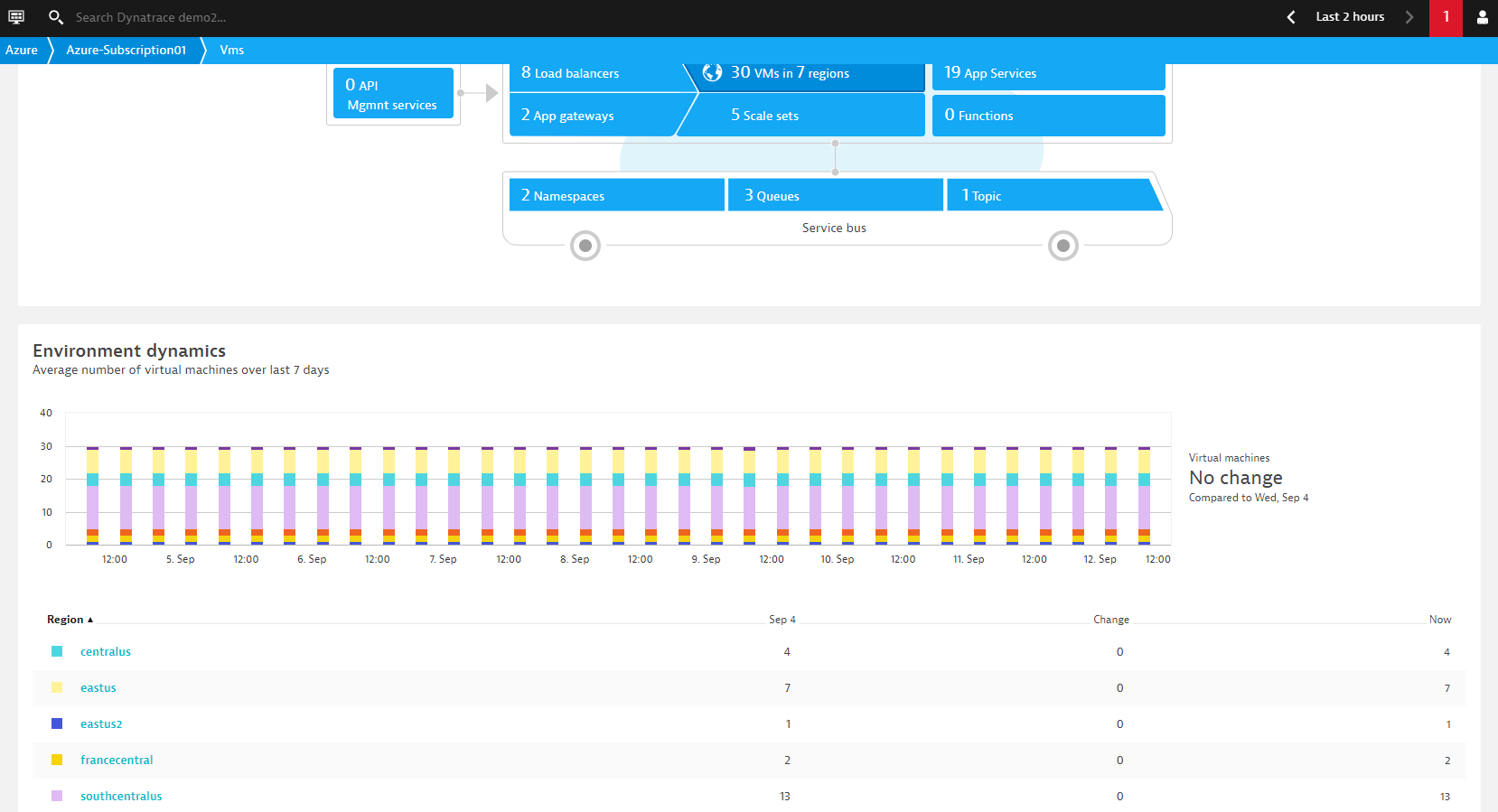


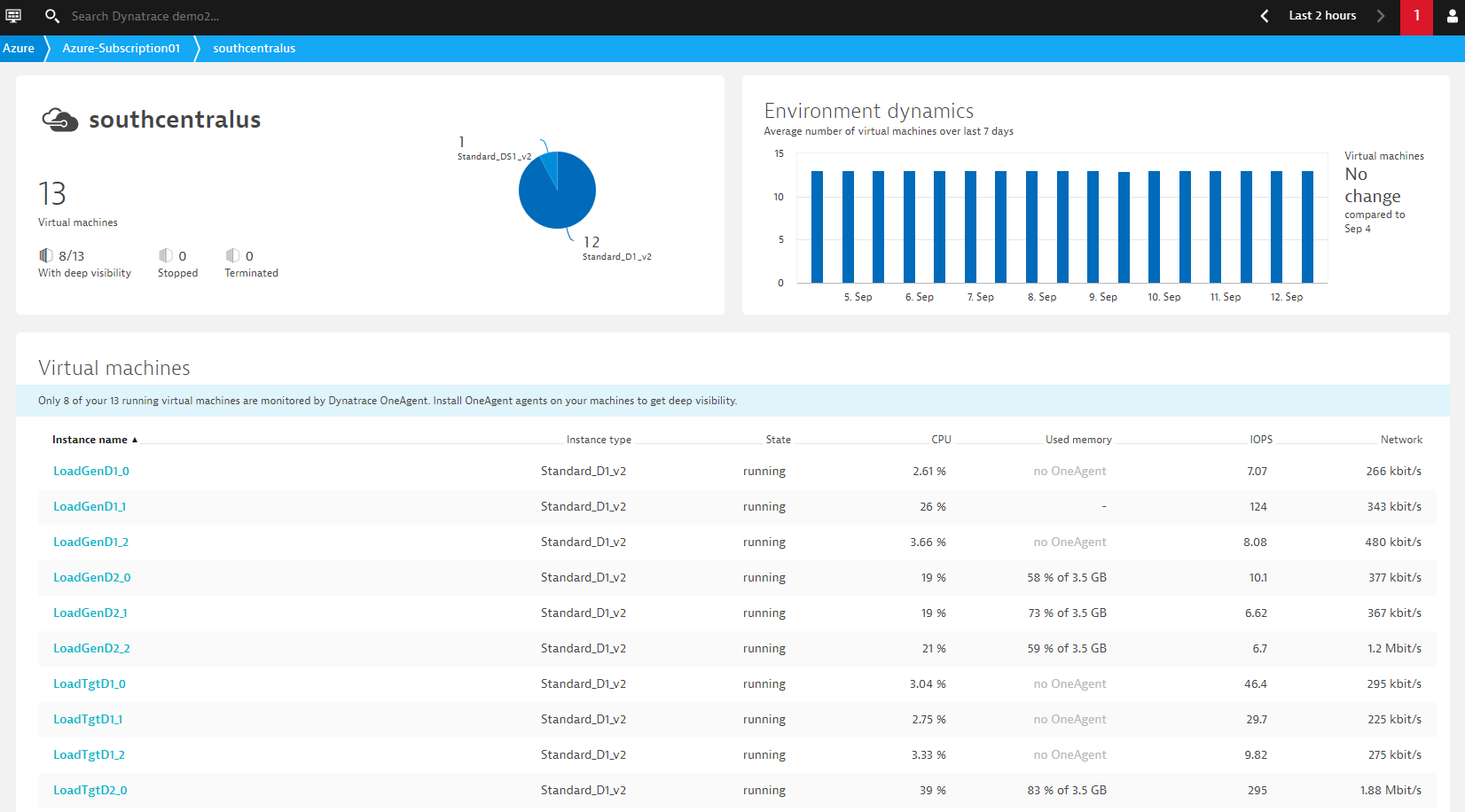




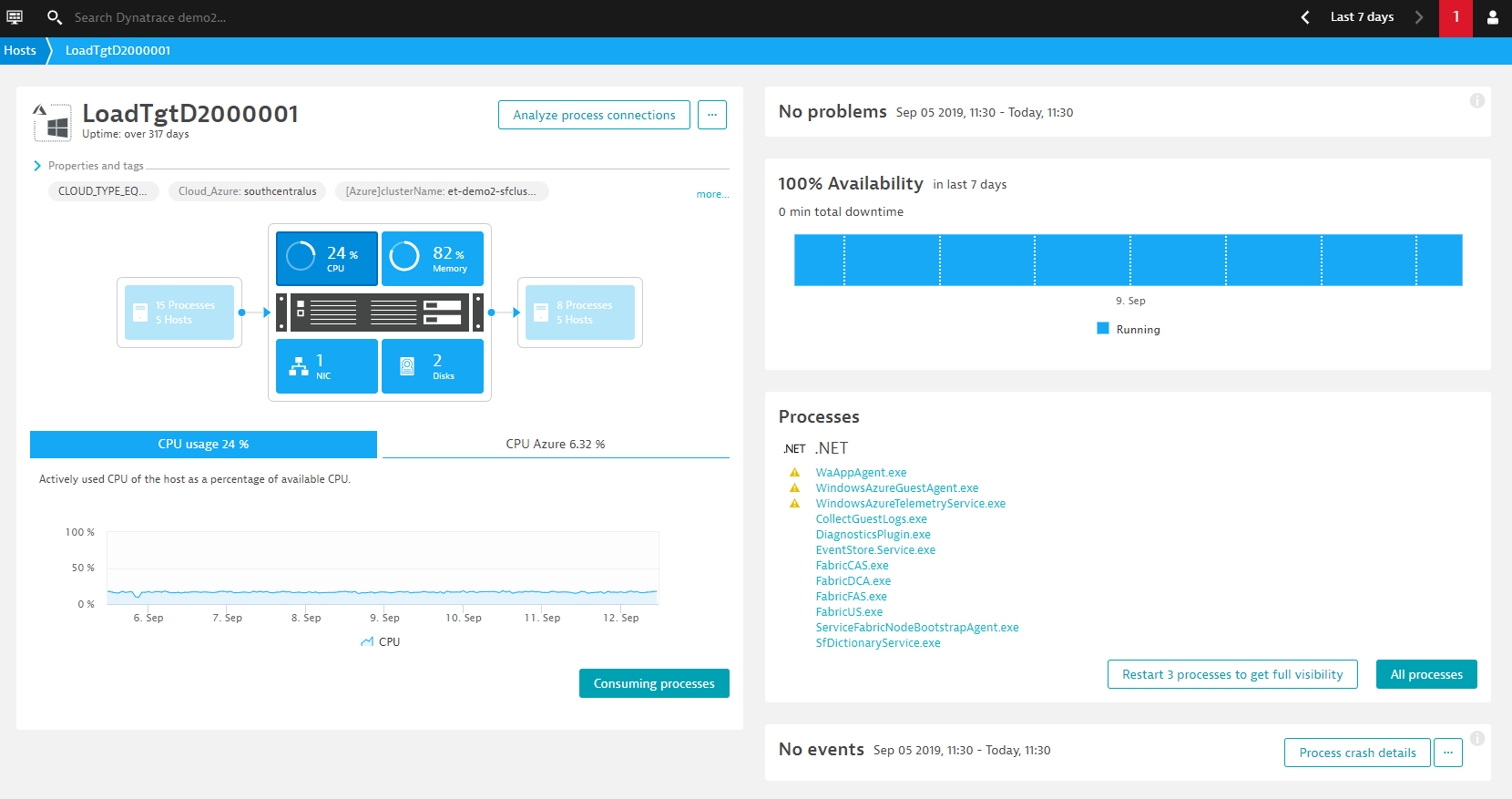


3. Using VM as an example, select the desired region to further drill into the details. The infrastructure status could be viewed in a centralized place.



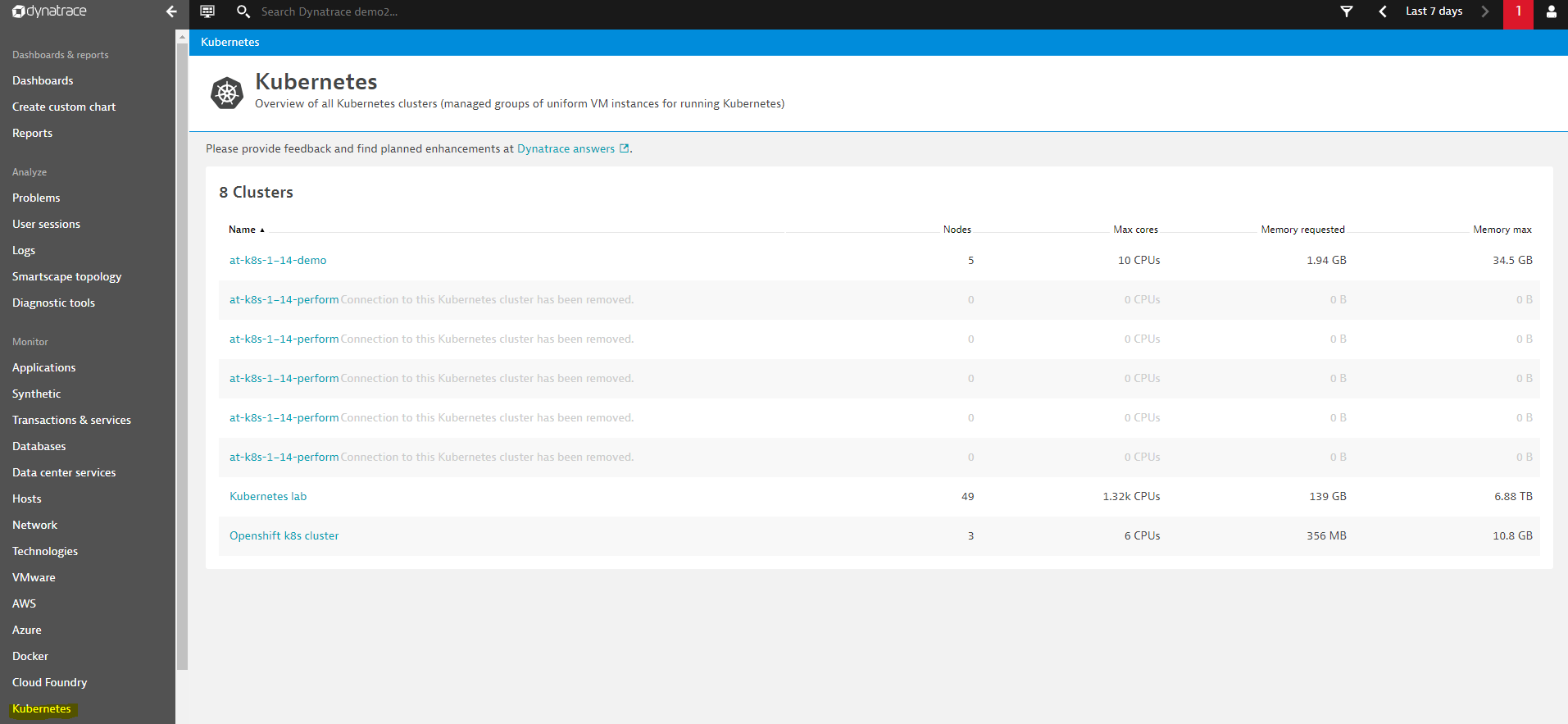


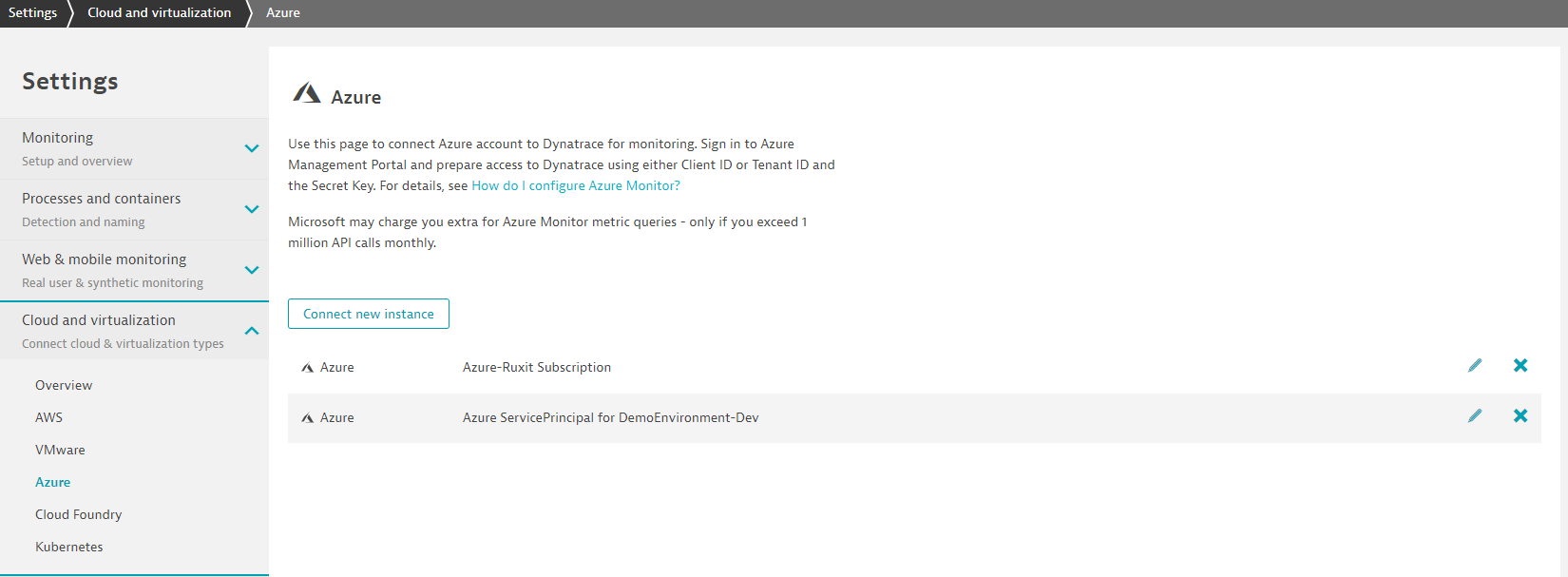
4. Further drill into a particular instance for more visibility. The infrastructure, problem, process status within the selected timeframe will be displayed. In this example, the result implies the selected host has 100% availability with no problem occur during the last 7 days.



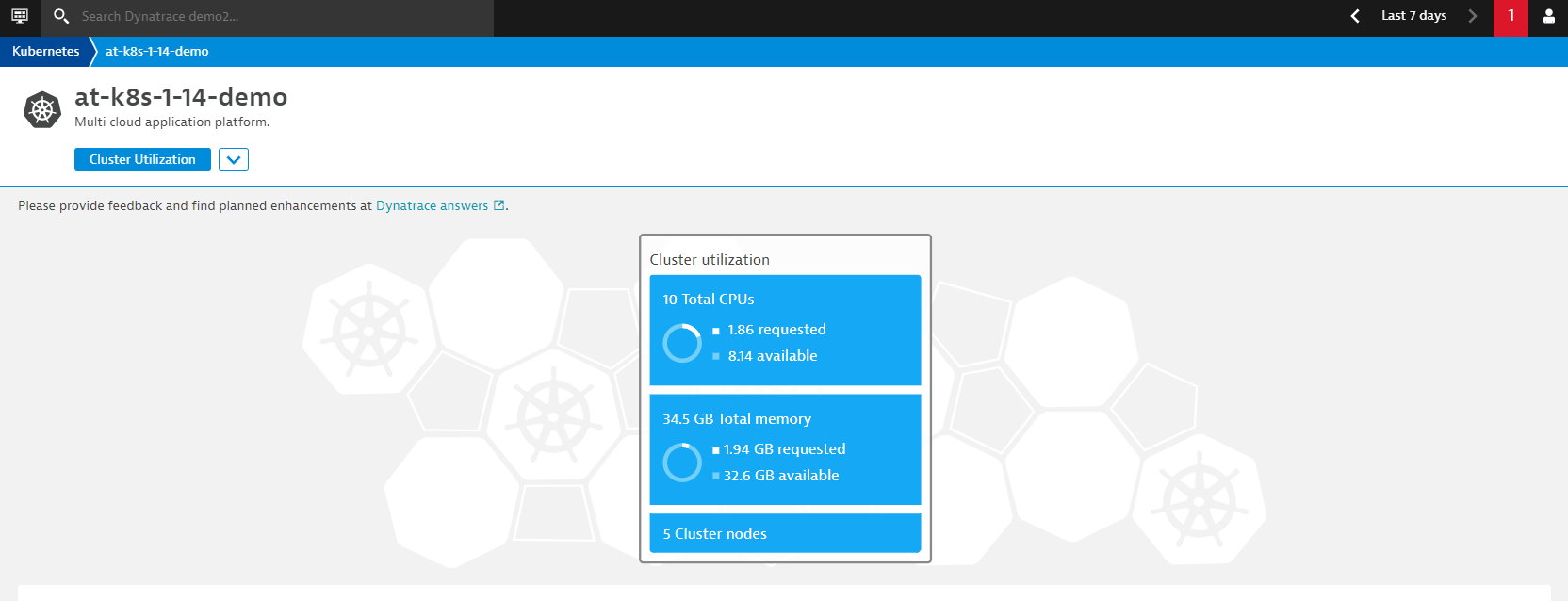
### Kubernetes Section

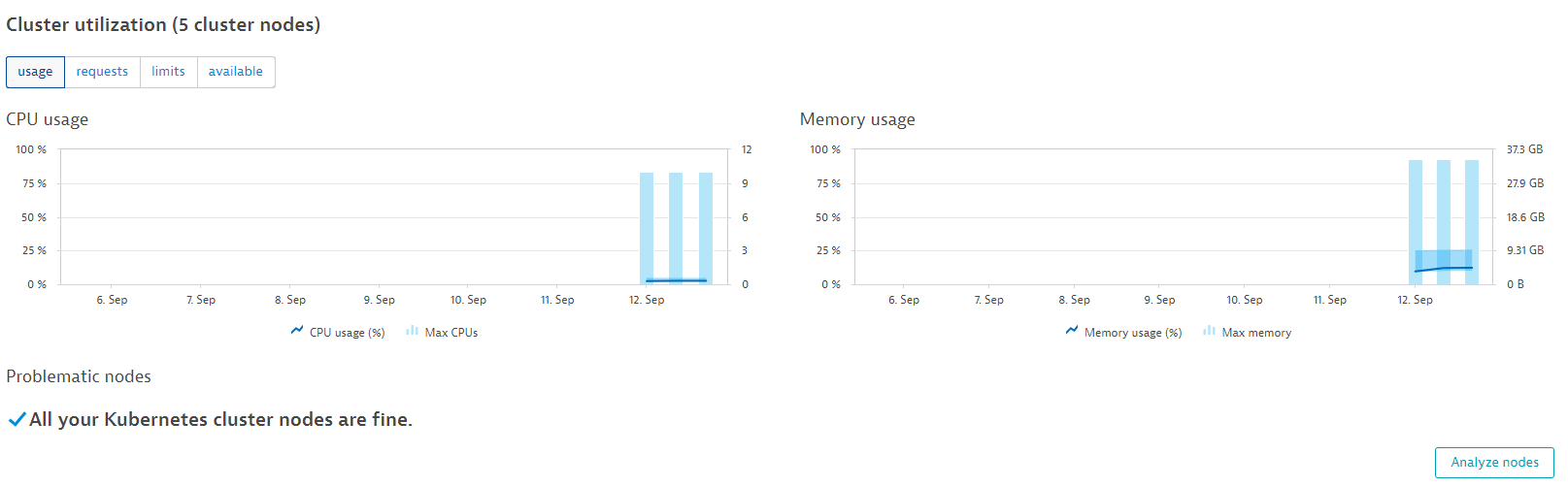
1. Kuberentes monitoring feature is also natively supported by Dynatrace, and Azure Kubernetes Service (AKS) is already fully integrated with the k8s monitoring feature. Select Kubernetes in the left menu bar to start the investigation. The monitored clusters are listed in a summary page, click into the cluster for more details.

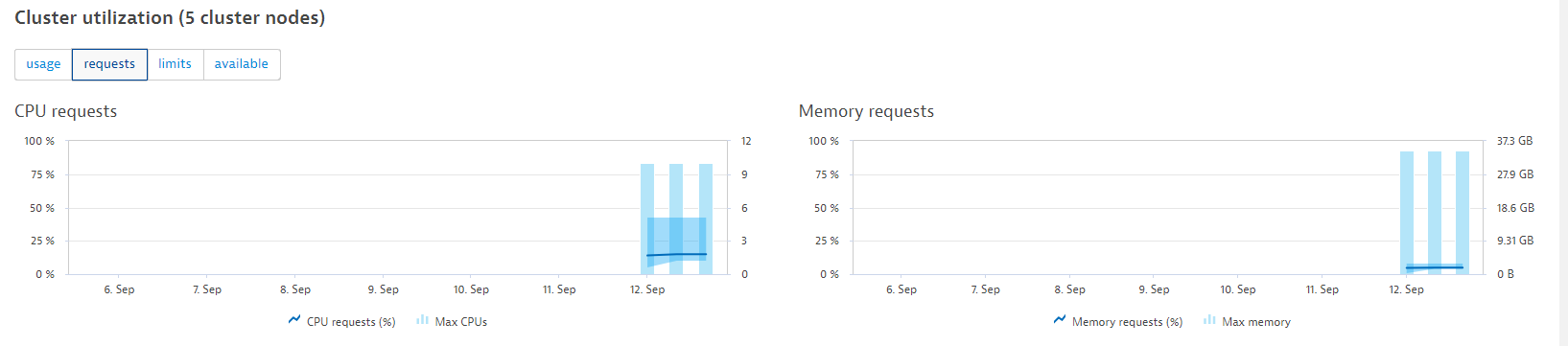


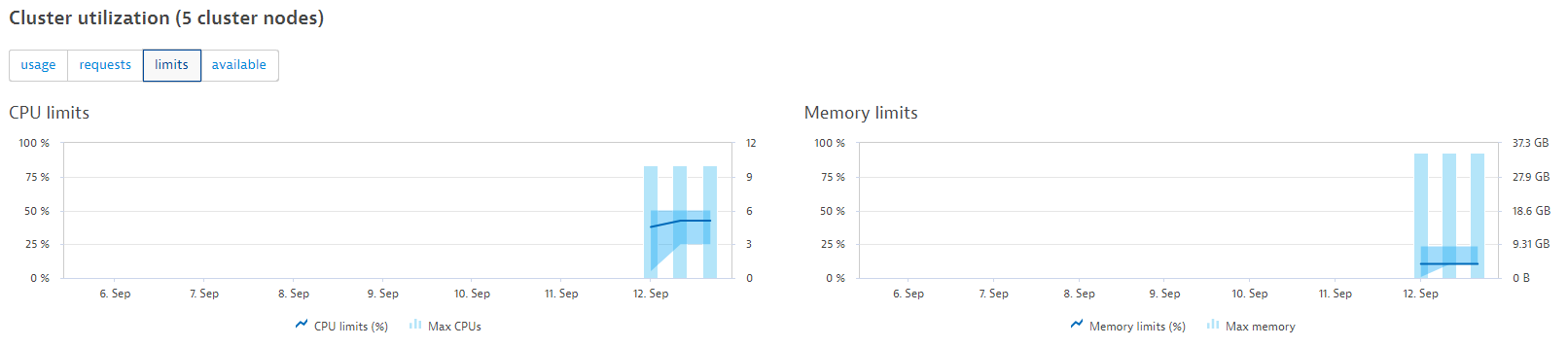


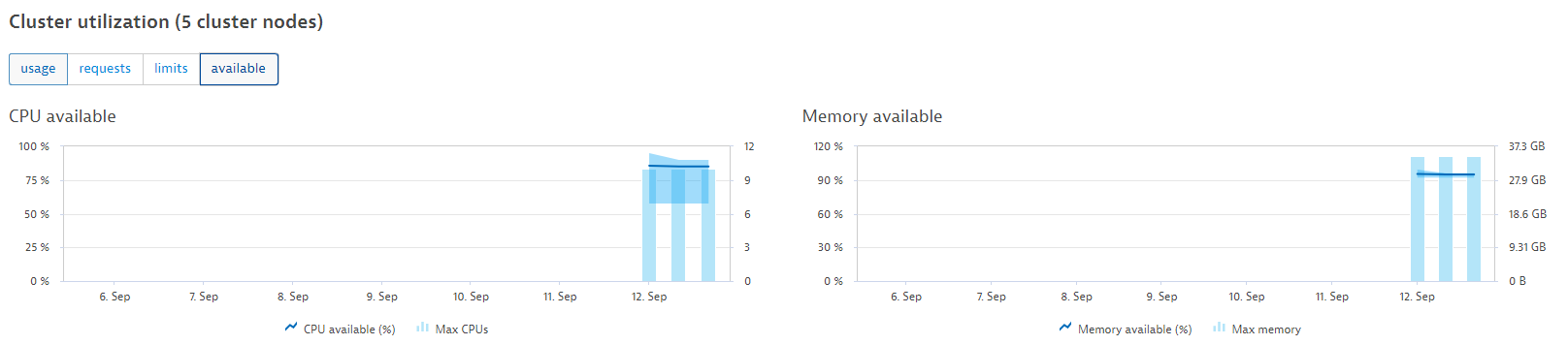
2. Similar to the Azure section, users can review the resource utilization in a summary page.



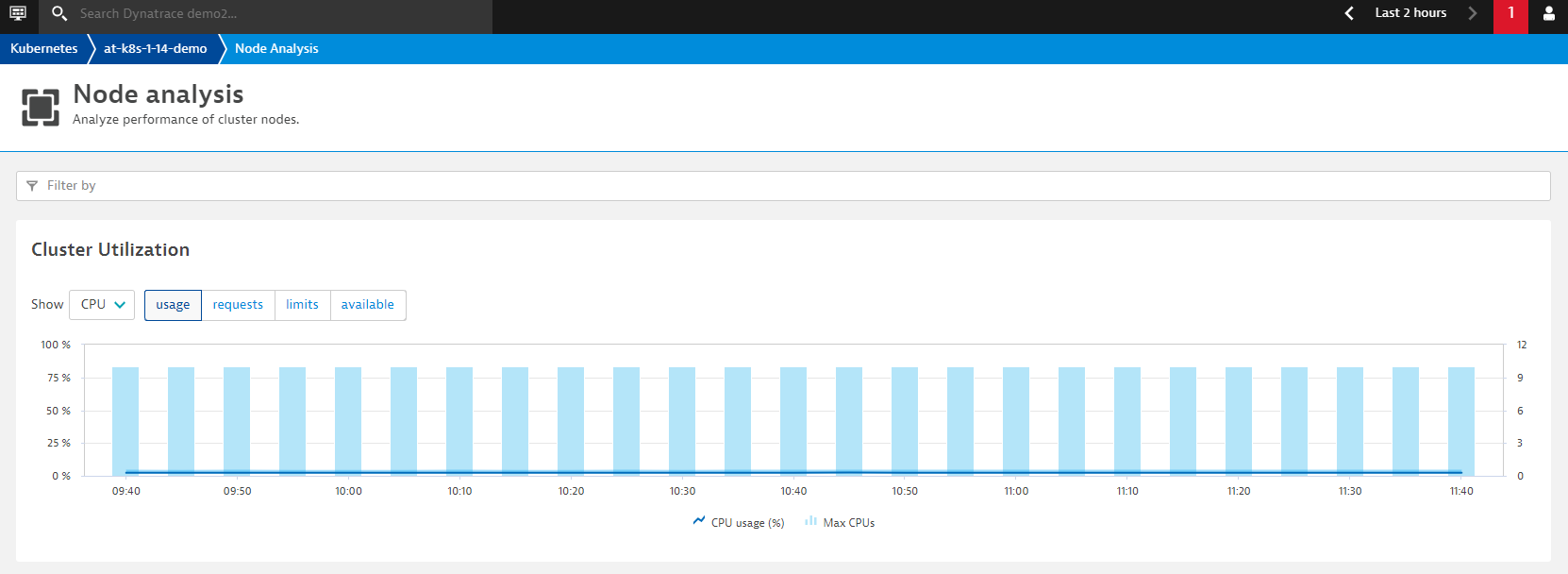


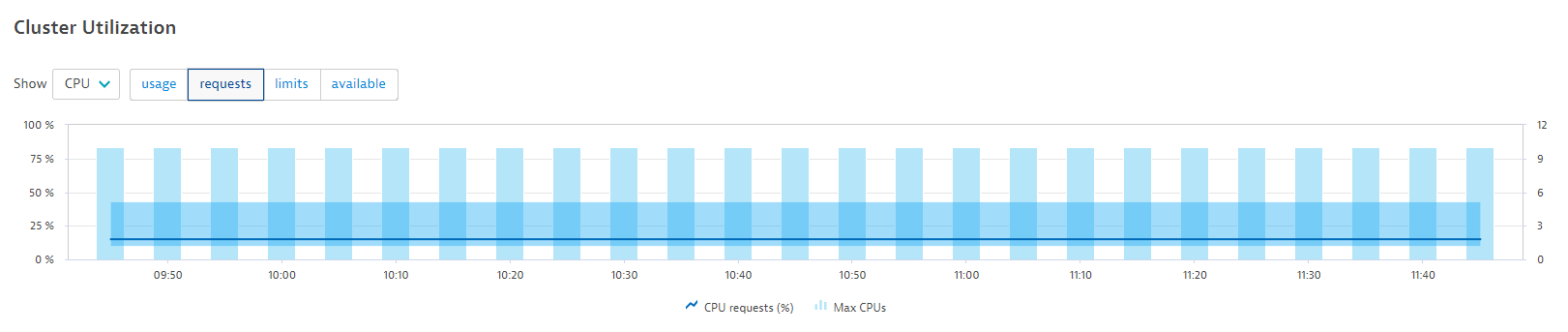
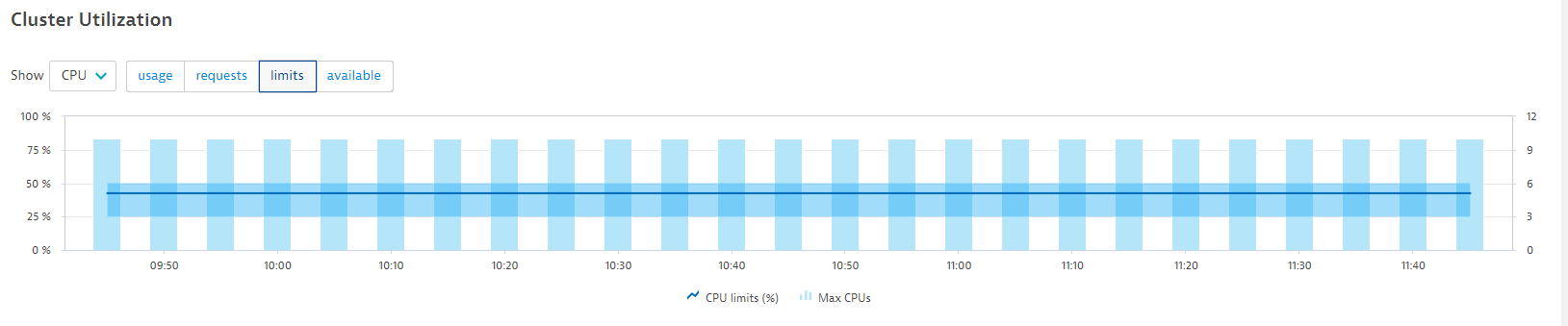


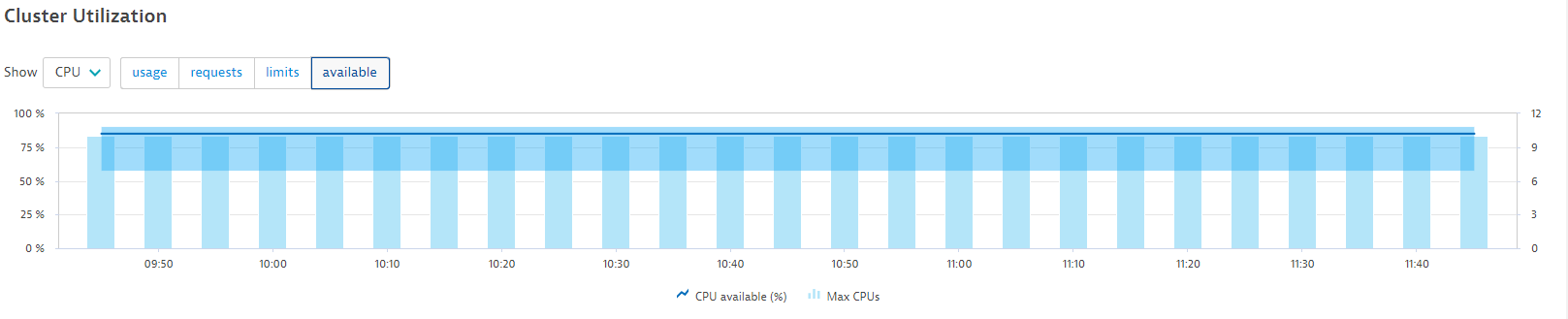


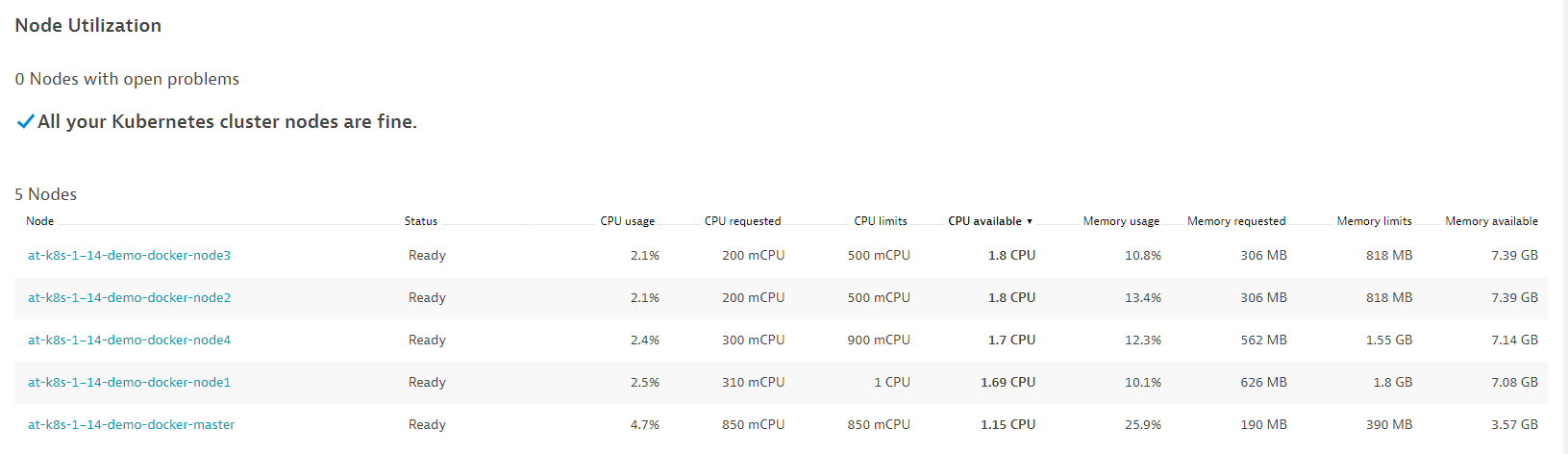


By clicking into the Analyze Nodes, the overall node details will be displayed.

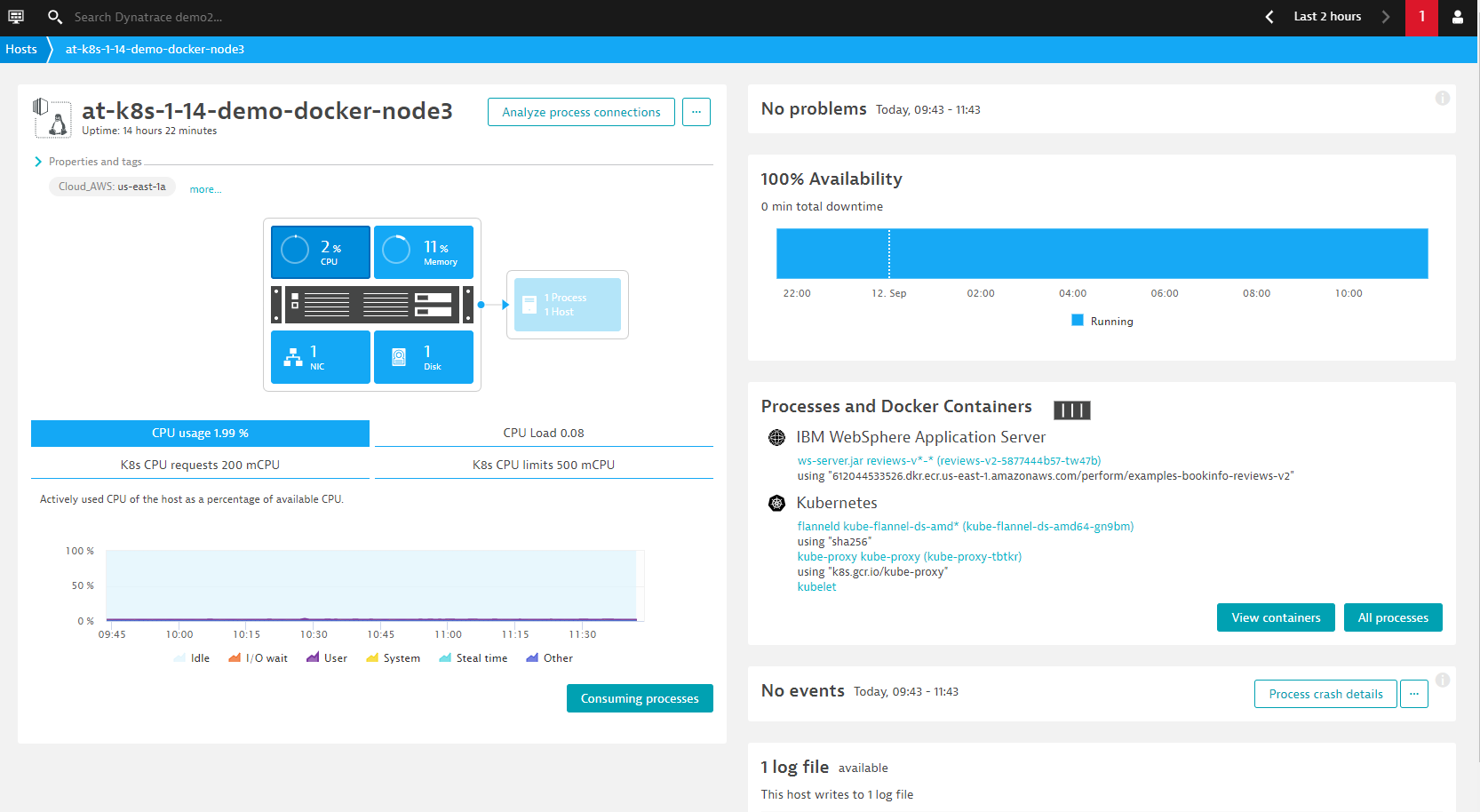




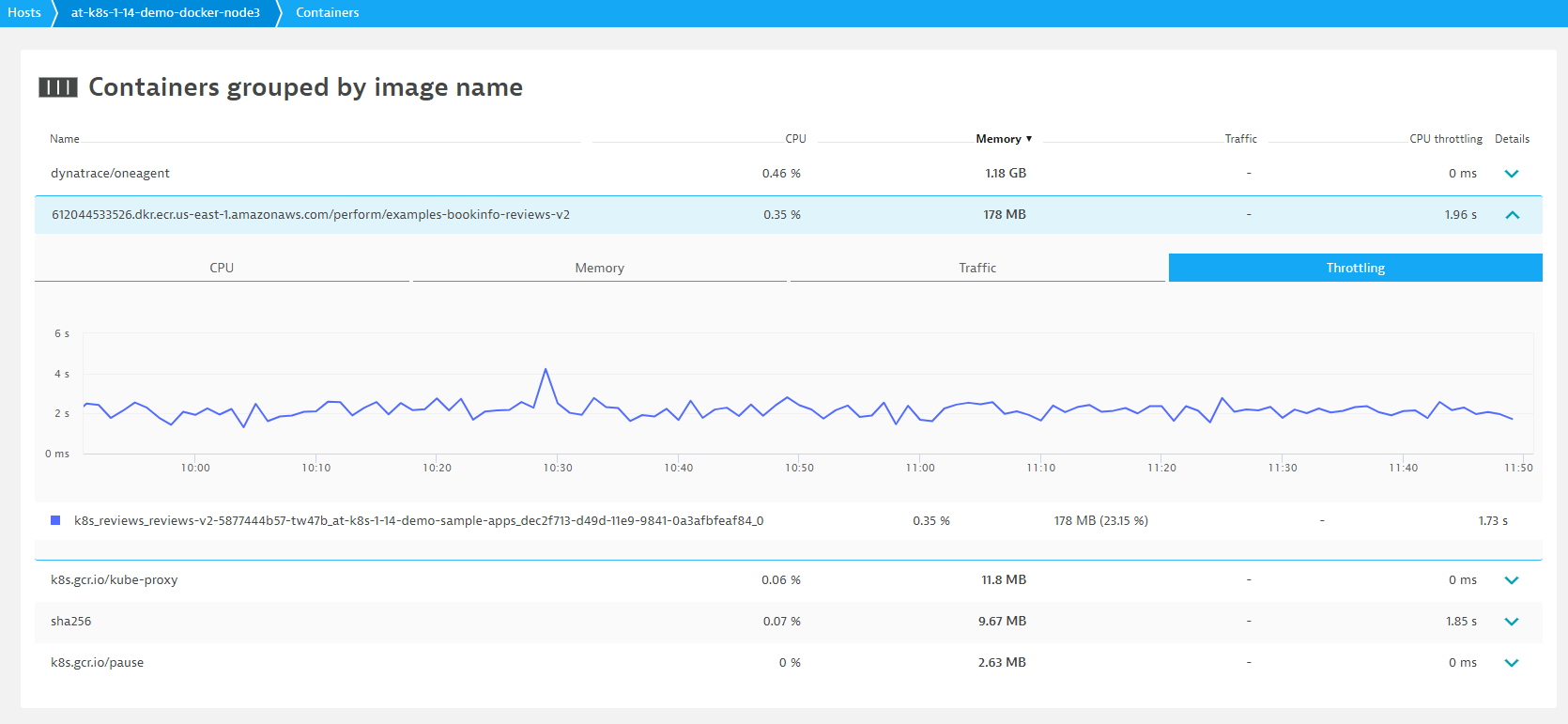


4. Node by definition, is a worker machine in Kubernetes, it is known as a VM or physical machine depending on the cluster nature. By clicking into particular node, users could review the infrastructure, process and problem status of the node unit, similar to what we have reviewed in the Azure VM section. In this example, it represents that the infrastructure resource consumptions of the node are normal with 100% availability, and no outstanding problem related to the processes and Docker containers inside the node over last 2 hours.



5. Users can further drill into the container for more visibility.



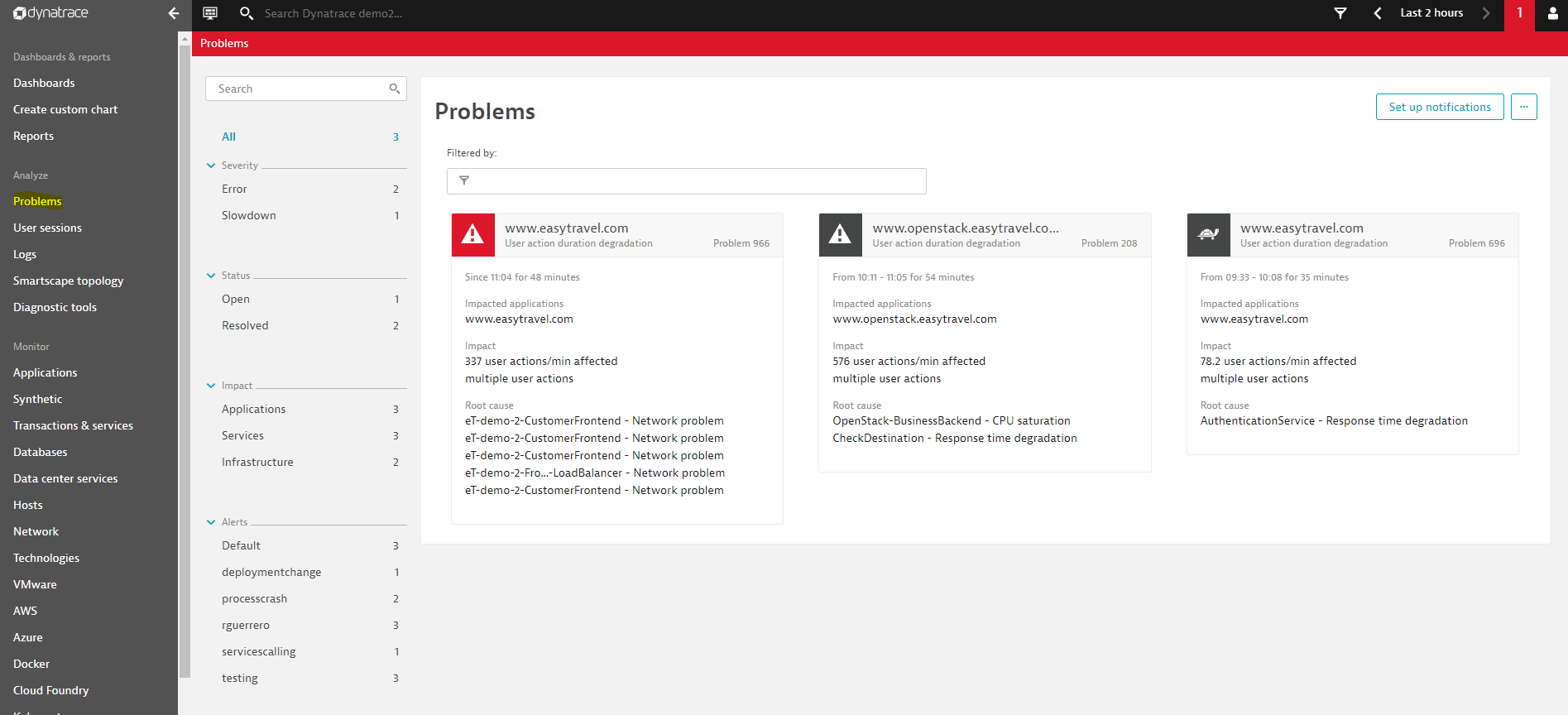


## General Problem Diagnostics

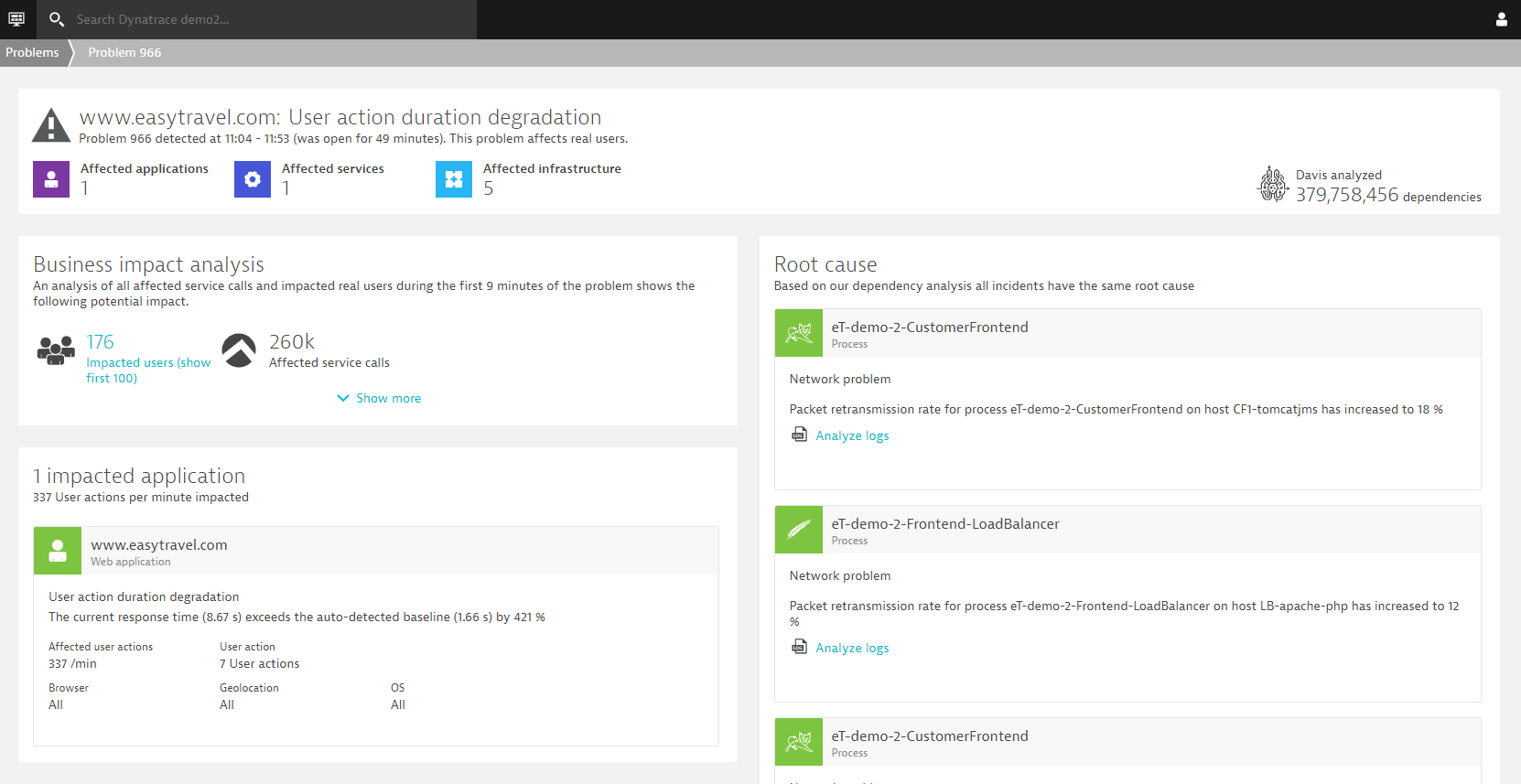
There are several dimensions to investigate a single problem, this section will provides a generic practice for users to gain a full-dimension visibility over the issue.

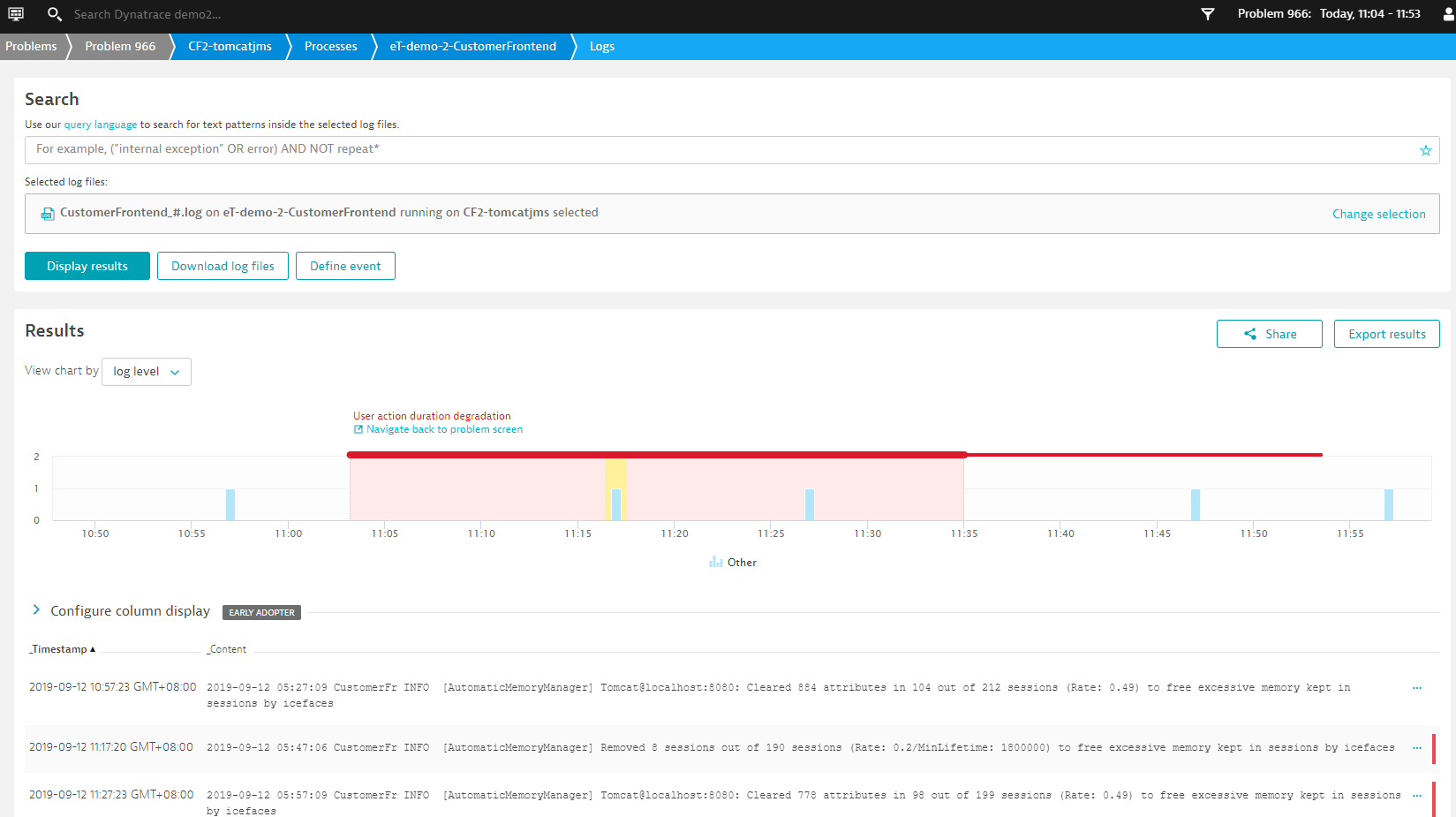
### Problems

1. The most common approach to investigate a problem in Dynatrace is to rely on the automatic problem discovery feature. If there is any outstanding problem within the selected timeframe, a red index will be displayed at the top right corner. User can click into the red index, or select Problems in the left menu bar to visit the problem overview. Click into the problem to obtain more insights.

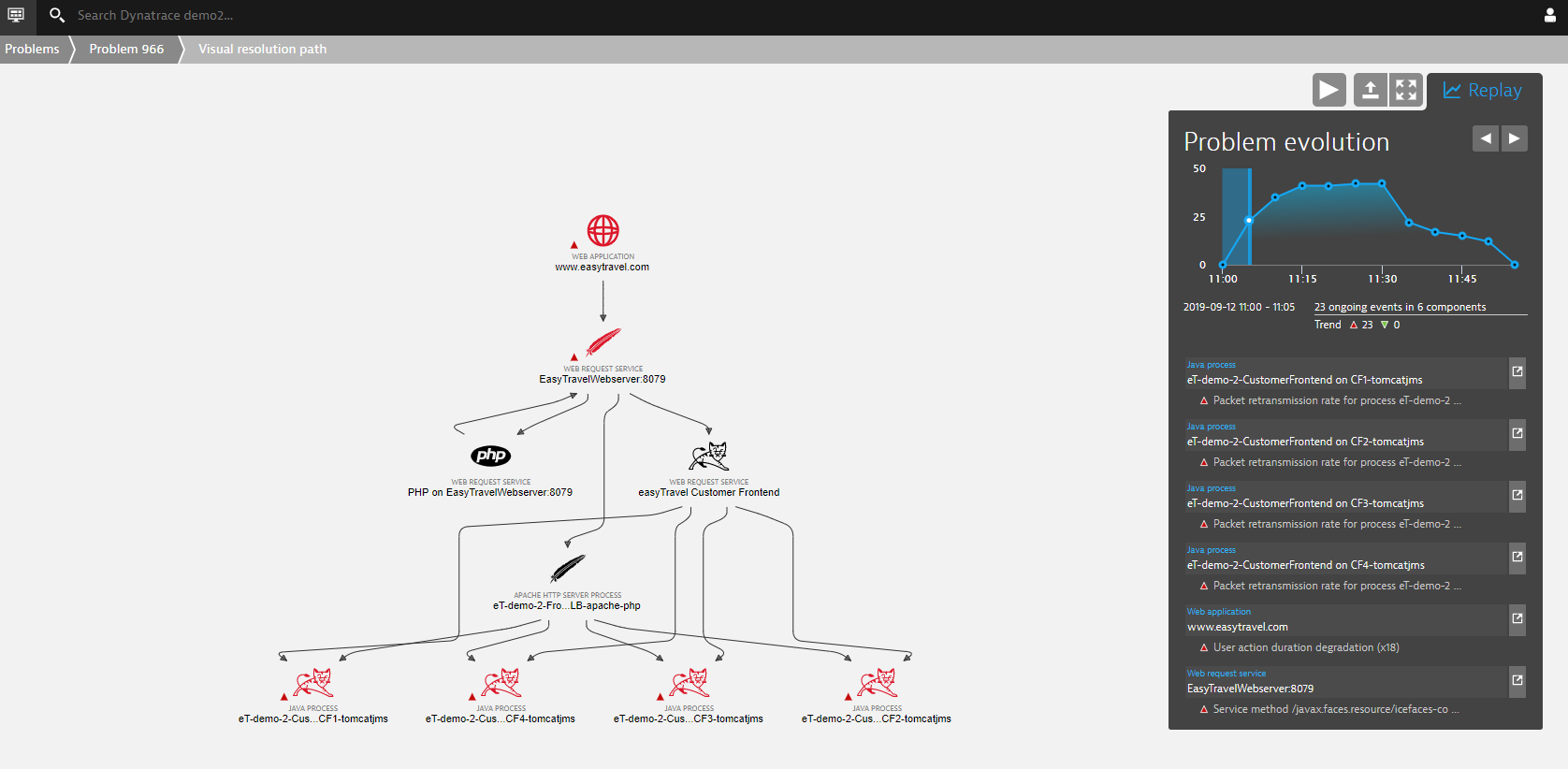


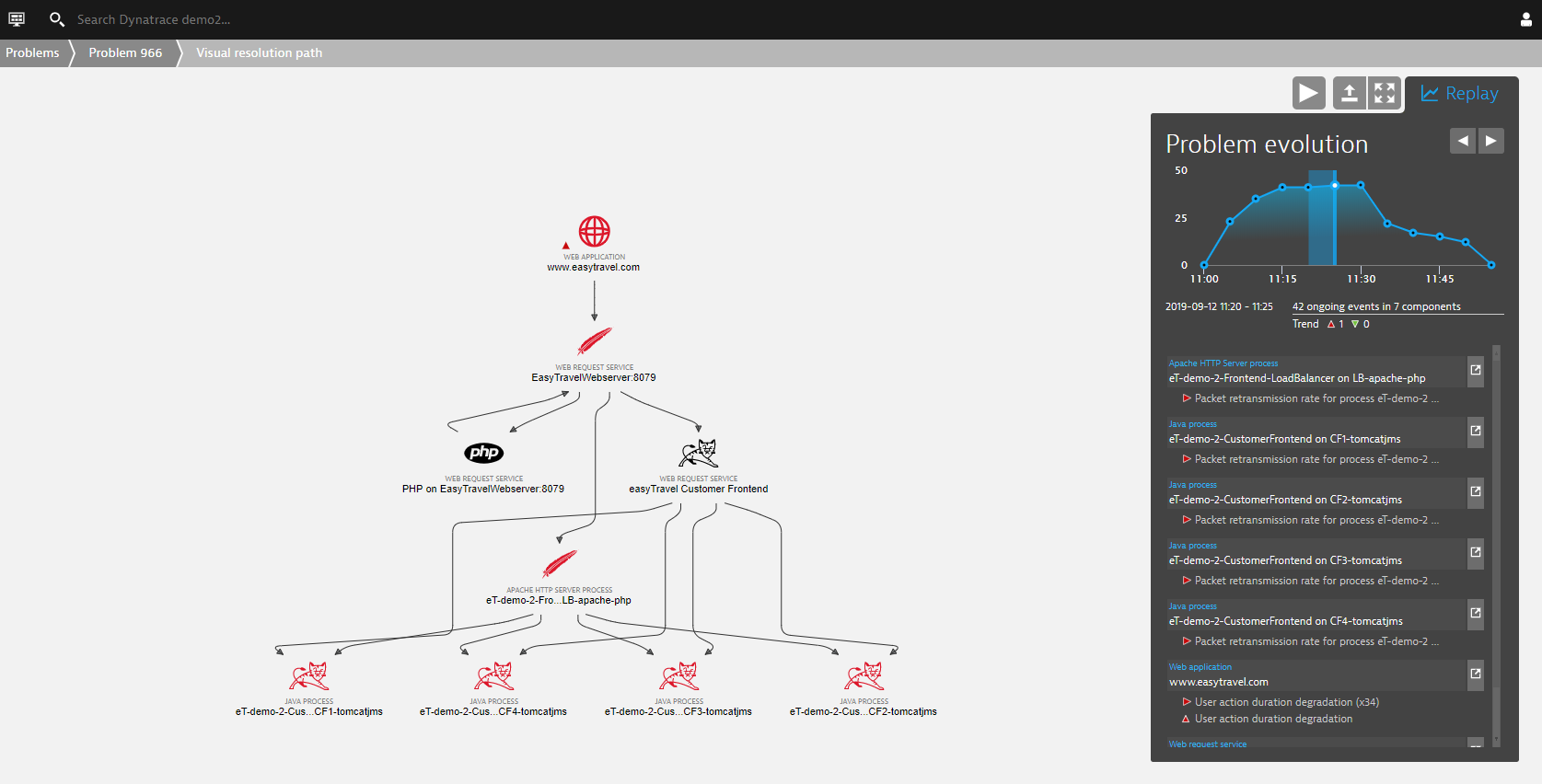
2. By clicking into the particular problem, the impact of the issue (i.e. affected user section, affected service calls, affected applications), as well as the root causes are discovered and detected automatically by Dynatrace’s AI brain - Davis. Review the root cause description one by one, or even drill into the log for deeper analyzing.

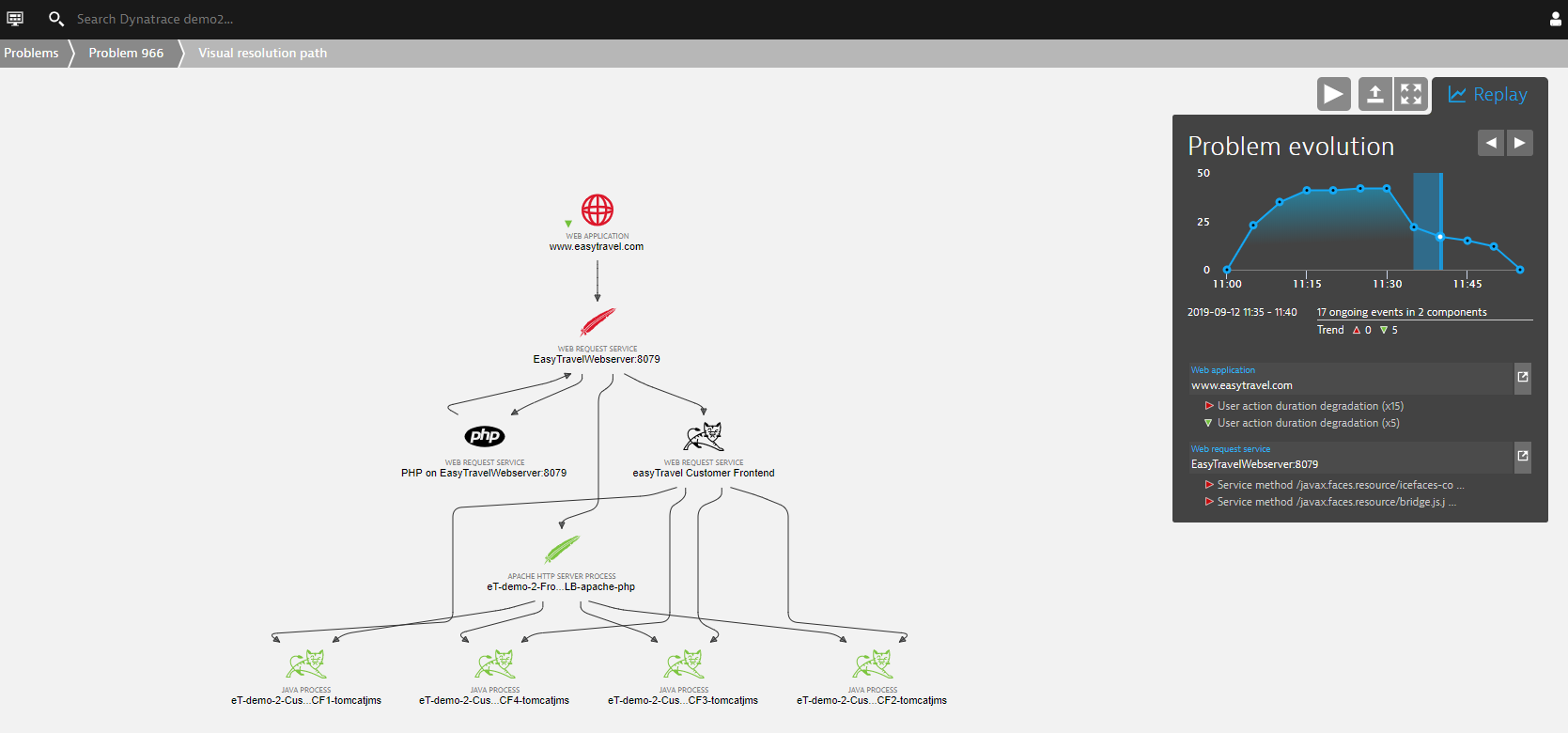


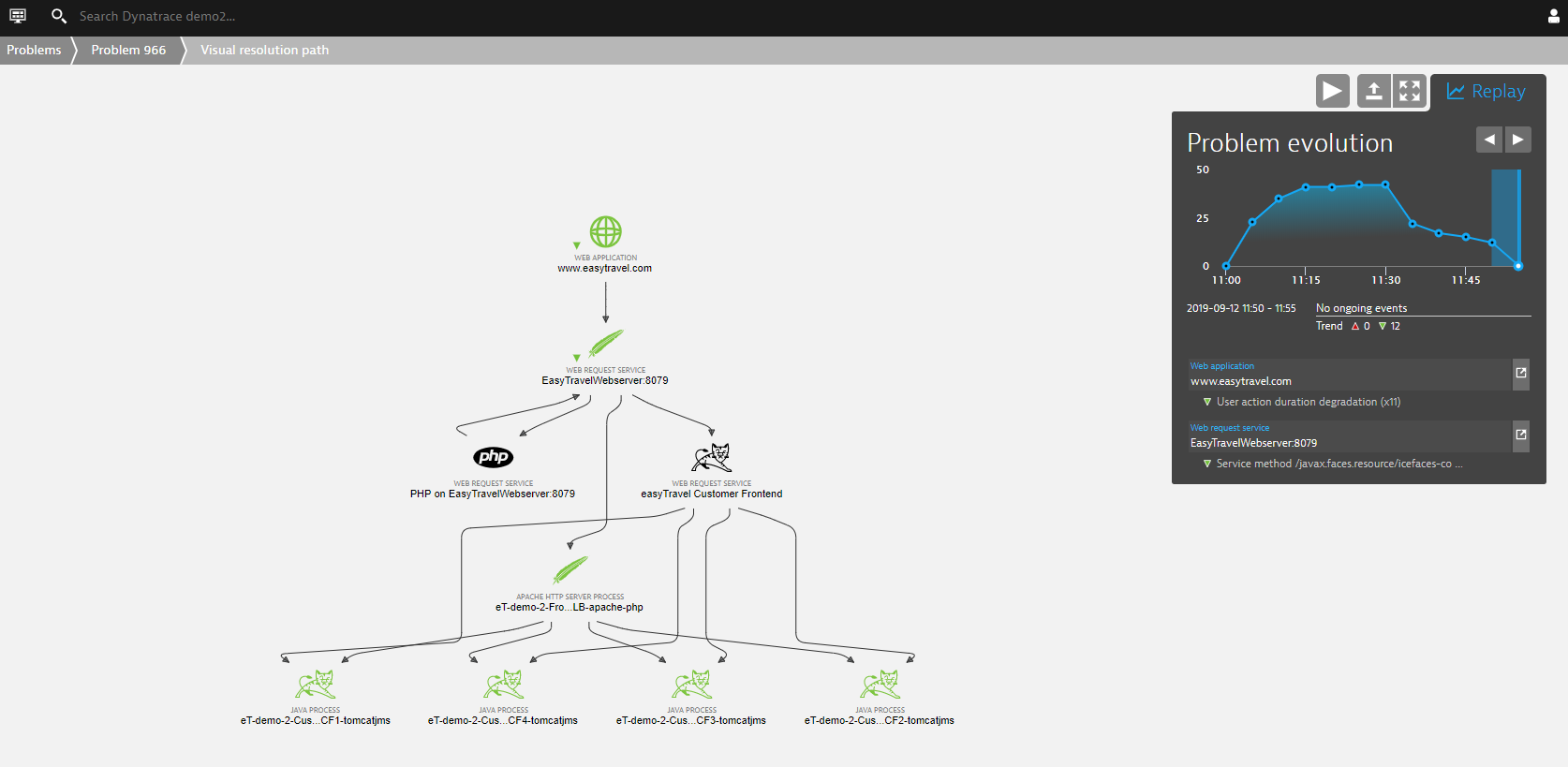


3. User can also make use of the Visual resolution path to playback the problem occurring sequence to investigate the root issue contributing component.





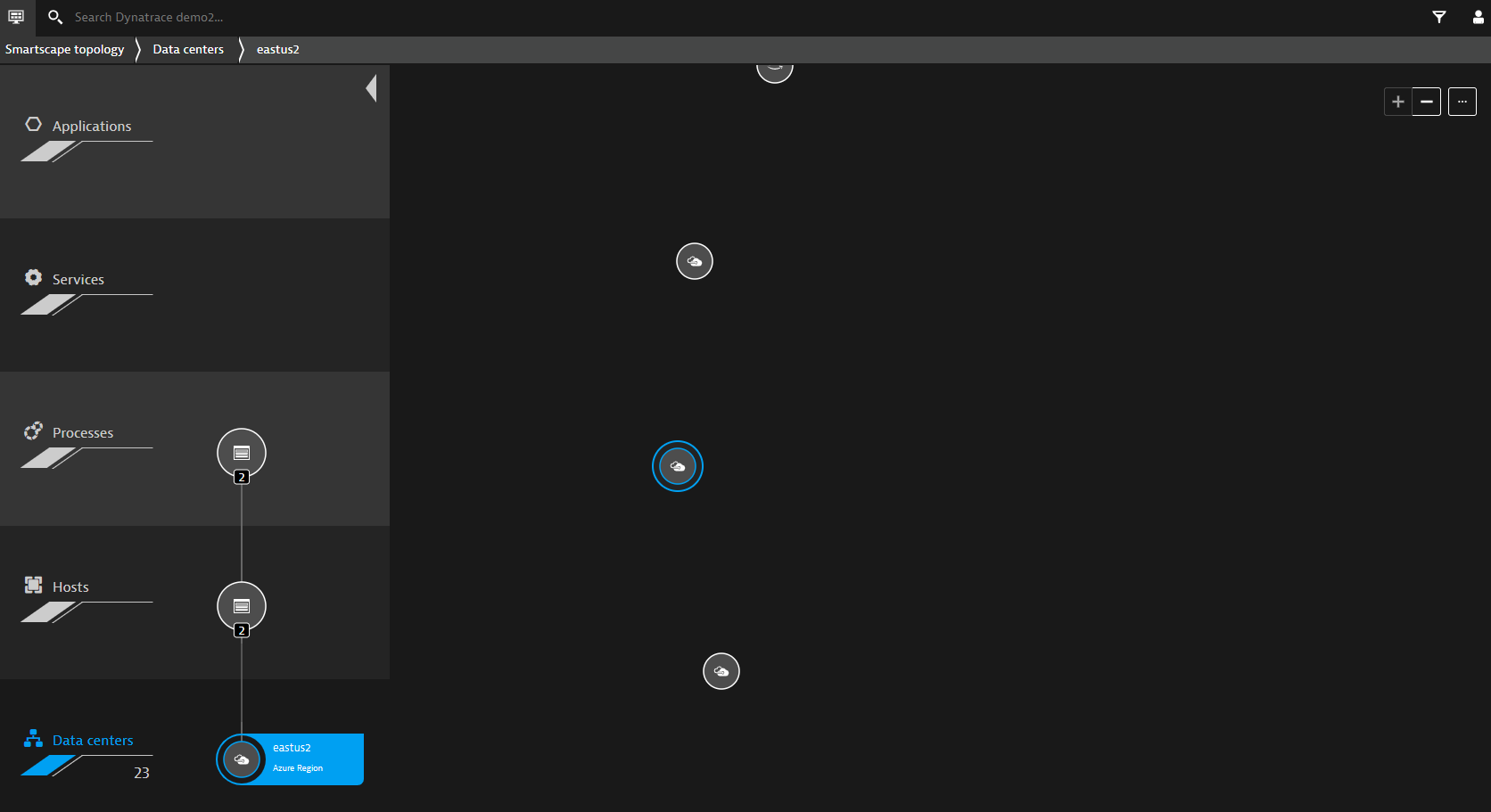


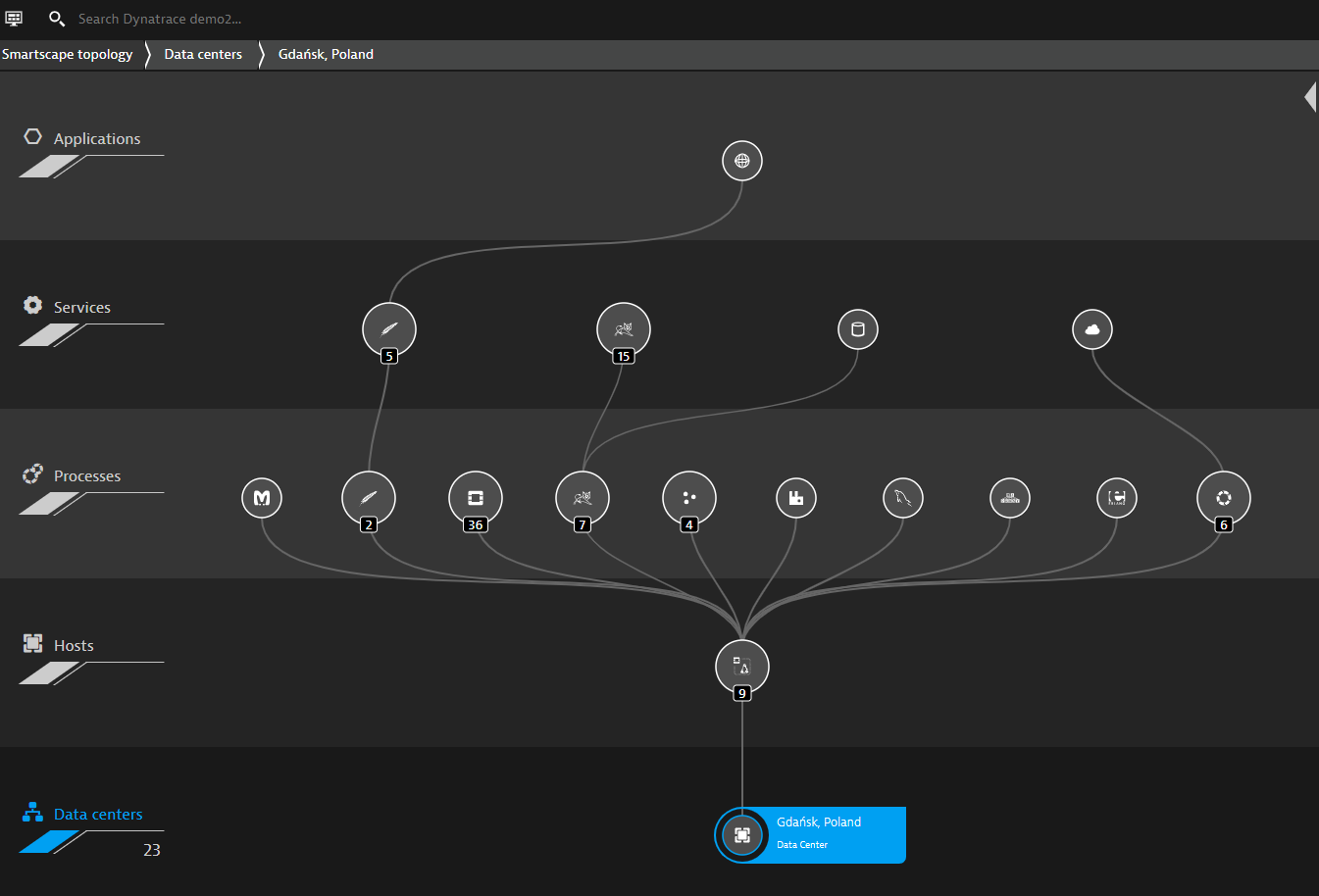
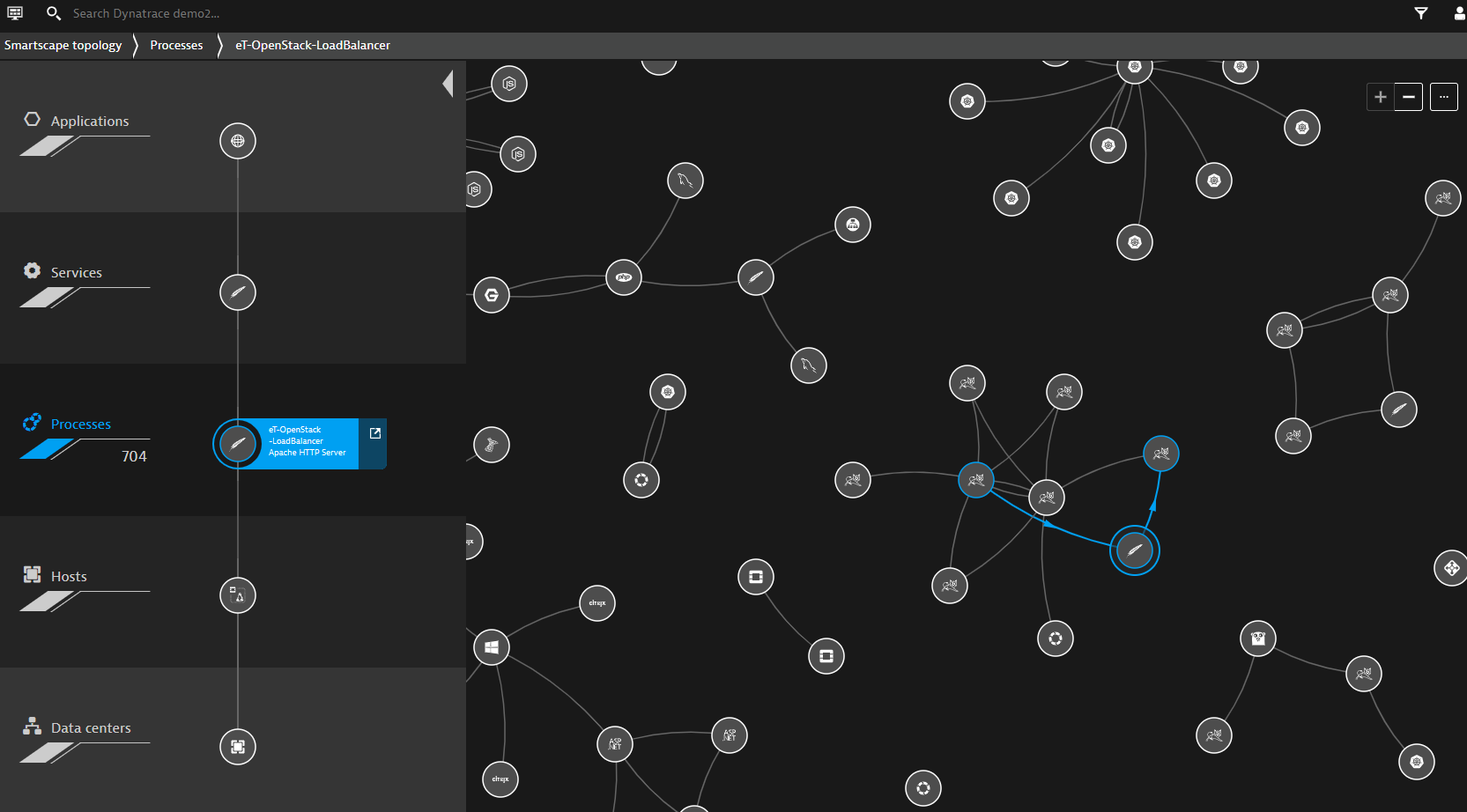


### Smartscape topology

If users expect to have a full stack visibility on the overall component linkage and relationship, select Smartscape topology in the left menu bar. The smartscape provide the full-stack description to the monitoring infrastructure. The linkages are done automatically by Dynatrace agent, from Data centers to Hosts, from Hosts to Processes, and down to Services and Application levels. With a visualized central point of view, users could easily review and identify their infrastructure correlation between components, it will be useful for troubleshooting the dependency issue and planning for the resource migration or updating. If any issue occurred, the hotspot shown in the direct will also turn red, use could then drill into the highlighted hotspot for further investigation, from a full-stack perspective.





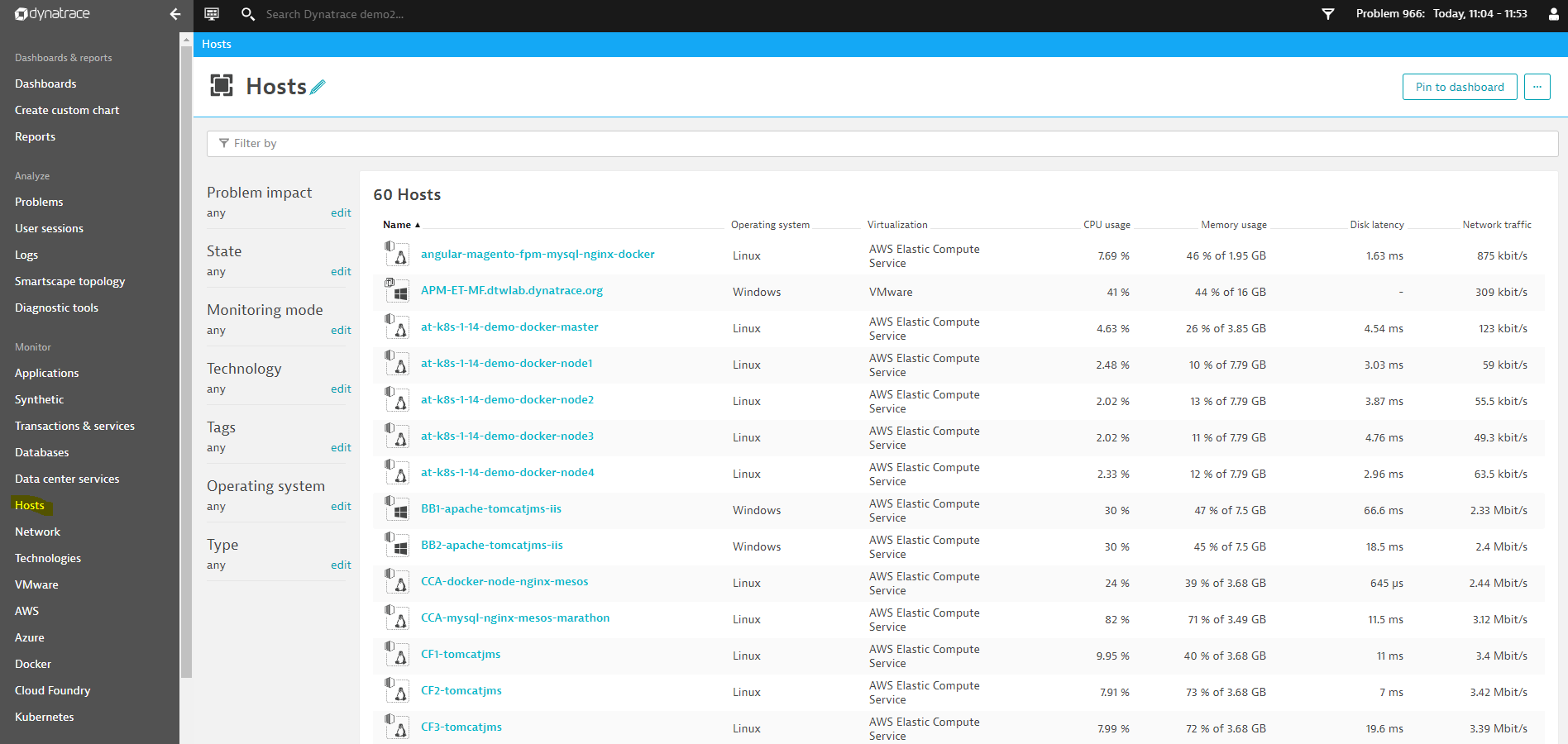
  


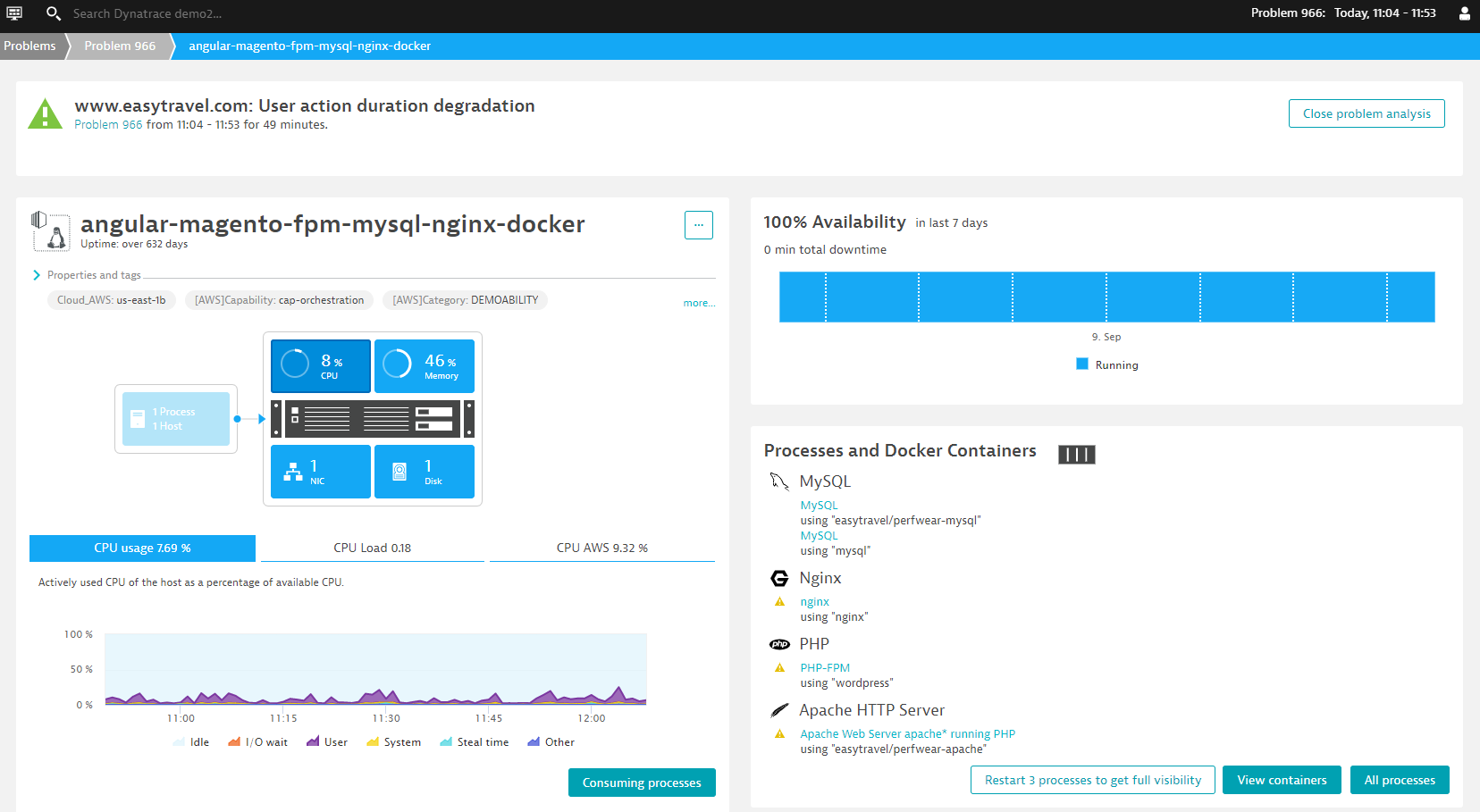
### Other monitoring perspectives

Different users would have different expertized domains, an example would be the skillset and focus difference between the Application Developer and Network Administrator. Dynatrace provides different monitoring perspectives to ensure different expertized users could efficiently obtain the information they know or they want, to facilitate the troubleshooting process as a whole.

#### Hosts

By selecting Hosts in the left menu bar, the information in host level will be displayed. Drill into it for more host-related details.





#### Databases

By selecting Databases in the left menu bar, the information in host level will be displayed. Drill into it for more database-related details.



