A Broadcom Company
technologies

White Paper

Establishing Effective APM Implementations in Kubernetes Environments

Introduction to APM and Kubernetes

Executive Summary

Application performance management (APM) is more challenging than ever, due in no small part to the rise of platforms like Kubernetes. And yet little information is available on how to contend with these challenges. This white paper aims to close that gap. In the next few sections, we will discuss what modern APM entails, and how it is being upended by the demands of platforms such as Kubernetes. Later sections will focus on different aspects of setting up and managing Kubernetes in a way that simplifies APM.

How APM Is Being Revolutionized: Microservices, AIOps, and Beyond

To begin, we will discuss how APM has changed as a result of the advent of platforms such as Kubernetes. APM has existed as a discipline within the IT industry since at least the early 2000s, when tools like Wily first appeared on the scene.

Yet despite being a relatively *old* type of technology (by IT standards, at least), the APM ecosystem is witnessing massive innovations. You might even say that we are in the midst of an APM revolution. In the next few sections we explain why.

Traditional APM

When APM first appeared, the fundamentals of the technology were pretty straightforward: you collected data, analyzed the data for anomalies or patterns that could signal a problem, then manually addressed the problem. First-generation APM tools excelled at collecting data and alerting IT teams when something appeared to be off, based on preset configurations that defined what was *normal* and what was not.

Little about this changed fundamentally for the first fifteen years or so of APM's existence. The tools became a little more user-friendly. Alert management grew more sophisticated, reducing instances of alert fatigue. The APM ecosystem grew much more diverse, with a variety of commercial solutions appearing alongside the original open source monitoring platforms. And many APM platforms shifted from an on-premises architecture to a SaaS model.

Yet despite these changes, the core functionality of APM remained the same: your tools collected data, analyzed the data, and sent you alerts. The types of environments managed by APM did not change much either. Until about five years ago, most teams were still monitoring the same types of virtual and physical servers and monolithic applications that they had been using since the early 2000s.

The APM Revolution

Yet all of this began to change starting around the middle of the last decade, which touched off an APM revolution. This was the result of a variety of factors.

Microservices and Complex Infrastructure

One spark for the APM revolution was the need to manage the performance of more complex types of applications and infrastructure. As Docker containers (which appeared in 2013) and Kubernetes (which debuted two years later) gained massive adoption, and as development teams migrated to microservices architectures, the nature of application environments and the infrastructure hosting them changed radically.

Today's environments consist of many more layers and moving pieces. It is no longer enough to collect just basic monitoring data, such as whether a server is up or how long an application takes to respond.

Instead, APM tools must now be able to collect and interpret a broad range of data from all of the different components of an environment, understand the relationships between different data points, and interpret them collectively to generate meaningful insights. The root cause of a performance problem in one microservice could lie within another microservice, or deep within one of the infrastructure layers that hosts the microservice. In a Kubernetes environment, a failure could be caused by any of the various components of Kubernetes: the API server, the Kubelet agent, the master node, or a worker node.

Traditional APM tools cannot handle that type of complexity, which is why modern APM has shifted toward microservices-aware tools that can map the complex relationships between the various parts of an application and the infrastructure on which it runs.

Dynamic Baselining

A related factor driving rapid innovation in APM today is that we are living in a new normal in which there is no normal, at least when it comes to application monitoring and management.

The profiles of modern application environments change so rapidly that it has become impossible to establish a baseline of ordinary activity and use it to detect anomalies. The number of instances of an application (or a microservice within an application) could fluctuate constantly in response to changes in demand. So could the resources allocated to host infrastructure or the network traffic patterns within the environment.

Amidst this constant change, APM requires a new approach based on dynamic alerting thresholds that can distinguish between unremarkable changes and those changes that signal a meaningful issue. Conventional APM tools were not designed to do this, but next-generation solutions are embracing the need for dynamic interpretations of software environments.

AIOps

AIOps, too, has become a key force behind the APM revolution. Although the term AIOps did not appear until 2016, you could argue that AIOps existed before then in practice, since IT operations teams have long used data analytics to help provide visibility into software environments. However, what changed in recent years is that AIOps has grown much more sophisticated, with tools that are capable not just of reacting to events or making suggestions about how IT engineers should act, but also of automatically remediating problems.

Along with that change has come important new opportunities for APM tools. In the past, the role of data analytics within APM was limited mostly to helping to parse log files in order to draw out relevant patterns or anomalies. Today, data analytics can do much more by making prescriptive recommendations and driving automated decision-making on the part of AIOps tools.

Put another way, AIOps means that APM platforms are gaining critical new types of functionality that extend data-driven automation across APM workflows. Instead of just using analytics to interpret monitoring data and leaving the rest of the work up to engineers to perform manually, AIOps is enabling more automated and systematic approaches to APM.

Approaching Your First Kubernetes Installation

In many ways, the hardest part of Kubernetes is getting started. This section offers beginner-friendly guidance on the essentials of launching a Kubernetes cluster in a way that will set you up for long-term APM success.

Getting Started with Kubernetes: Overview

When you start learning about Kubernetes, the first thing you need to do is set up a local development cluster for testing configurations and deployments. You usually begin by installing a Minikube in a virtual machine, which simplifies the learning process and lets you work out example scenarios.

However, when you are ready to go to the next level (which is a production installation of a full Kubernetes cluster), you will need to start over from scratch. And depending on your business requirements, you will need to make sure to include controls for securing the confidentiality, integrity, and availability of the system.

This is a daunting process, and it requires reading the documentation and keeping detailed records. For those of you who still need an extra helping hand, we have compiled a list of recommendations for setting up your first production installation of Kubernetes.

Let’s get started.

Follow the Official Docs First

It is very tempting to download and bookmark dozens of tutorials for installing Kubernetes and managing configurations. While this is helpful, it is also dangerous, because these guides sometimes contain mistakes. In addition, these tutorials can be biased and promote tools that might not be suitable for your needs.

You should always consult the official Kubernetes docs, since they give you the complete and correct installation steps for each supported version of Kubernetes. Once you have mastered these steps, then you can explore the other resources.

Start Minimal

Start with the minimal recommendation for master nodes (three master nodes and three control planes) using the stacked etcd topology. The setup is straightforward, and it is not difficult to configure.

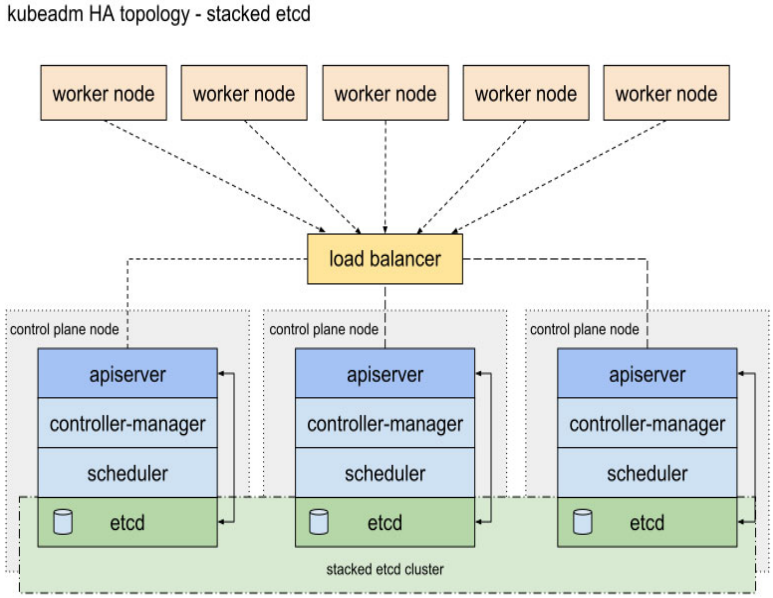
Start by installing etcd on each node and generating the certificates for encrypted communication. Once the etcd cluster is operational and healthy between the three nodes, then you can set up the K8s cluster using the kubeadm tool. You will have to provide an initial configuration, and the easiest way to get started is to inspect the examples from the following outputs:

```
$ kubeadm config print init-defaults
```

```
$ kubeadm config print join-defaults
```

You will have to provide the IP for the etcd nodes and the load balancer address. The following figure depicts the example topology:

Figure 1: Example: kubeadm HA topology with stacked etcd.



As long as the nodes can reach each other, all you have to do is install kubeadm on each one and then join them together. The tooling will take care of the rest.

For setting up the load balancer, you can use a variety of options, such as the one described here: https://www.keepalived.org/ or here http://www.haproxy.org/. This may be the most complex step in the installation process, but it sums up the whole experience.

Once you have deployed the master nodes, the hardest part is behind you. Now you can add the worker nodes and test that the orchestration is up and running.

Use the Best Hardware and Software for OS

When you pick the OS and the hardware for the master and worker nodes, do not be stingy. You will need a commercially supported OS like Ubuntu or RedHat with the latest Linux Kernel so that you can find support easier and faster in case of software failures. When choosing hardware, you should allocate high frequency CPUs, more CPU counts, and more memory for faster operational performance. Later on, you can use taints and tolerations to maximize usage.

You need at least 2GB of RAM and two CPUs for each node. It is best if you double or triple that to 8GB of RAM and four to eight CPUs, and that is only for the basic services. If you want to add monitoring or logging, then you need to add extra memory. You will also need more RAM and more CPUs, depending on the number of worker nodes you plan to deploy.

A fast SSD with RAID for the etcd server and a fast 10GbE network for low-latency node communication is a must.

Use Kops

Once you are proficient in installing a cluster using kubeadm, we recommend that you automate that process using Kops. Among other things, Kops makes upgrading easier, it has maintenance tools for production environments, and it can generate Terraform configurations for setting up clusters.

Terraform can considerably improve the process of automating the provisioning of Kubernetes clusters by using code that can be checked out in a version control system. Kops can deploy to AWS, GCE, and Digital Ocean, although Bare Metal support is best left to kubeadm.

Another option is to use the Cluster API. Although it is still in the experimental phase, Cluster API has support for Azure.

Approaching Your First OpenShift Installation

Once you have wrapped your mind around the basics of Kubernetes, and know how to deploy a simple cluster, you can move onto running more complex clusters based on platforms like OpenShift. This section offers an introduction to setting up OpenShift.

OpenShift Overview

Whether you have already settled on OpenShift as your potential Kubernetes platform or you have only seen it in industry announcements, the best way to know if it is for you is to try it out by deploying a full cluster. If you are a developer, however, this may be overkill. For basic validation, you can use CRC (Code Ready Containers), which is a single-node version that can be integrated into your local workflow.

When you deploy a full cluster with an offering as complicated as OpenShift, there are several things that you need to take into consideration. First and foremost, though, you have to understand what OpenShift is and what it does.

Red Hat OpenShift Container Platform is the leading enterprise Kubernetes distribution.

That is because it has been a part of the Kubernetes open source community from its inception, and it has built on Kubernetes by bundling all of the components that an enterprise needs to get started with containerized applications.

The OpenShift distribution starts with pure upstream Kubernetes and integrates additional open source projects into a supported platform. These products provide hundreds of additional function points including enhanced RBAC, Single Sign-on, CI/CD pipelines, a private registry, and a web console. The goal of OpenShift is to offer a consistently managed and provisioned platform that provides developers with the same experience no matter which type of infrastructure they use. This allows for seamless migration and growth from private clouds to hybrid and multi-cloud infrastructures.

Which Type of Installer?

OpenShift 4 has two types of installers (IPI and UPI), but they do not both work with every type of infrastructure. This may seem like it complicates things, but the goal is to provide the most flexibility to customers while simplifying the actual installation. This is a significant improvement over OpenShift 3, which had an Ansible-based installer that worked on every supported platform, but needed hundreds of flags to get it going.

In order to install OpenShift, you need to choose between an Installer-Provisioned Infrastructure (or IPI) and a User-Provisioned Infrastructure (or UPI).

The IPI option installs OpenShift in a very opinionated way. It creates and configures all components of the underpinning infrastructure from the virtual network to the load balancers.

The UPI option is used in two situations: the first is when the opinionated configuration does not support an internal requirement (such as the need to reuse an existing virtual network). The second is when your chosen platform does not have all of the options that are needed to fully deploy OpenShift. This is the case mainly with on-premises deployments in which VMware and bare metal do not have load-balancers or DNS servers to be configured.

Using an IPI installer on a cloud platform is by far the easiest way to get started.

Pick Your Base Infrastructure

Now that you understand the different types of installers, you can pick your base infrastructure.

There are supported IPI installers for the following environments:

- AWS
- Azure
- GCP
- OpenStack

There are supported UPI installers for the following environments:

- AWS
- Azure
- GCP
- KVM
- VMware
- Bare metal

Configure the Prerequisites and Run the Installer

Both IPI and UPI installers have prerequisites that need to be met before any installation can take place. For an IPI installer, this usually means creating the service account and assigning the correct roles so that it has access to do everything required to complete the installation.

For a UPI installer, there will be additional steps outlined in the documentation; these can be anything from installing the OS to setting DNS and load-balancer requirements. The UPI installer does not have much tolerance for mis-configurations. As a handy tip, you should make sure that all systems have host names assigned to their IP addresses and that the time is synced (etcd will work much better this way).

Once you have finished this step, download the installer for your infrastructure of choice, perform any additional configurations specified in the official documentation, and run the installer.

You will have a cluster up and running in about 45 minutes.

Day Two Operations and Next Steps

OpenShift has enough functionality built into it that most enterprises will be able to begin working with containerized applications as soon their cluster is up and running. However, OpenShift does not provide industry-leading solutions for every aspect of running in production. For example, the open source products that enable out-of-the-box monitoring and reporting have limitations when it comes to alerting, scalability, and creating a holistic view across all aspects of the platform. Red Hat relies on partners such as Broadcom to provide best-of-breed solutions to fill these gaps. AIOps from Broadcom enables better deep-dive analysis, a holistic view across multiple clusters, and machine learning capabilities to discover trends before they become problems.

DaemonSets in OpenShift and Kubernetes

For our final section, we will move onto a more complex topic: how to take advantage of DaemonSets, an advanced Kubernetes feature.

Why DaemonSets?

There are times when running the same application on every node in a specific set of nodes in a cluster, or even the entire cluster is required. Typically, applications that provide some kind of system management functionality require you to run the same application on every node in a cluster. Examples include storage processes, which use local storage devices attached to the nodes, and log consolidation and diagnostic services, which often feed data into an AIOps solution.

DaemonSets were created to address a specific need that was previously either handled with an application that would use a custom controller, or more often was managed at the system level outside of the Kubernetes cluster, either as a manifest under a kubelet's control if it was a container, or as a systemd-managed service. If you configure these system applications to run as part of the cluster, you can add or remove nodes and all the processes that they need to run will be automatically applied. This allows for the infrastructure components to be extremely generic and for you to reuse them across any cluster without the need for custom deployments scripts for each cluster.

The one real caveat with using a DaemonSet in the pure upstream Kubernetes is that its pods are created and scheduled by its own DaemonSet controller, which does not take into account any pod priority and preemption rules that the default scheduler may be using. When you use Red Hat OpenShift, it enables ScheduleDaemonSetPods by default, which makes DaemonSets scheduled by the default scheduler so that they respect these rules.

Creating a DaemonSet

Creating a DaemonSet is much like any deployment in Kubernetes. It is defined as a YAML file and installed with either `kubectl apply`, `kubectl create`, or by using `oc create` if you are using OpenShift.

In addition, if you are using OpenShift you need to disable the default node-selector configuration in the namespace you are planning to use for the deployment of the DaemonSet. Instead, use cluster-admin access.

```
$ oc patch namespace myproject -p \
    '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

The actual YAML file contains the API version and kind DaemonSet, and some metadata to define things such as the name of the DaemonSet. The additional information that goes under spec is essentially a standard pod deployment, only embedded under spec instead of being its own YAML file.

If there is a node-selector listed in the spec section in the YAML defining the DaemonSet, then the deployment will be limited to nodes with that label applied; if there is no node-selector, then the DaemonSet will run on every node in the cluster.

For more detail, see the code example on the following page:

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: broadcom-daemonset
spec:
  selector:
      matchLabels:
        name: broadcom-daemonset
  template:
    metadata:
      labels:
        name: broadcom-daemonset
    spec:
      containers:
      - image: openshift/hello-openshift
        imagePullPolicy: Always
        name: registry
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
      serviceAccount: default
      terminationGracePeriodSeconds: 10
```

And then you create the DaemonSet:

```
$ oc create -f daemonset.yaml
```

Finally, verify it is running on all nodes:

```
$ oc get pods -o wide | awk '{print $1" "$2" "$3" "$7}'
NAME                    READY STATUS  NODE
broadcom-daemonset-5tqwl 1/1   Running compute-1585532381-000003
broadcom-daemonset-s9j6k 1/1   Running compute-1585532381-000001
broadcom-daemonset-tc64g 1/1   Running compute-1585532381-000002
broadcom-daemonset-w75mg 1/1   Running compute-1585532381-000000
```

Additional Information

DaemonSets can be deleted at any time and can also be updated using the rolling update strategies like any other pod deployment.

Communicating with the pods in DaemonSets can be configured in four different ways. As with all pods, communicating with DaemonSets can be configured to be accessed by Node IP with a known port, which is less flexible than using either a headless or regular service. The final way to configure communication with the pods is to make it push-only, which is when the pods in the DaemonSet only send information to other pods in the cluster. This communication configuration is common when the DaemonSet is collecting and aggregating data, such as log files.

For more information detailed information on DaemonSets, you can visit multiple sites, including the official sites for Kubernetes documentation and Red Hat OpenShift documentation.

Conclusion

Kubernetes is a complex platform, so complex that it is revolutionizing the requirements that modern APM tools must address. To master APM for Kubernetes, you must understand not only how the APM landscape has changed since the advent of containers and microservices, but also how Kubernetes itself works; what its major components are, and what implications they have for monitoring and management.

In offering a basic overview of these topics, we hope this white paper helps you get started in learning how APM and Kubernetes intersect. For additional guidance, and to learn how the APM solution from Broadcom can help you cost effectively manage and scale your Kubernetes workloads, visit our Kubernetes monitoring page or contact us to set up a personalized conversation.