KUBERNETES: THE ULTIMATE GUIDE TO MASTER KUBERNETES

First edition. September 11, 2020.

Written by Clint Neal.

KUBERNETES

The Ultimate guide to master kubernetes

Clint Neal

Disclaimer

All erudition contained in this book is given for informational and educational purposes only. The author is not in any way accountable for any results or outcomes that emanate from using this material. Constructive attempts have been made to provide information that is both accurate and effective, but the author is not bound for the accuracy or use/misuse of this information.

Foreword

First, I will like to thank you for taking the first step of trusting me and deciding to purchase/read this life-transforming eBook. Thanks for spending your time and resources on this material.

I can assure you of exact results if you will diligently follow the exact blueprint, I lay bare in the information manual you are currently reading. It has transformed lives, and I strongly believe it will equally transform your own life too.

All the information I presented in this Do It Yourself piece is easy to digest and practice.

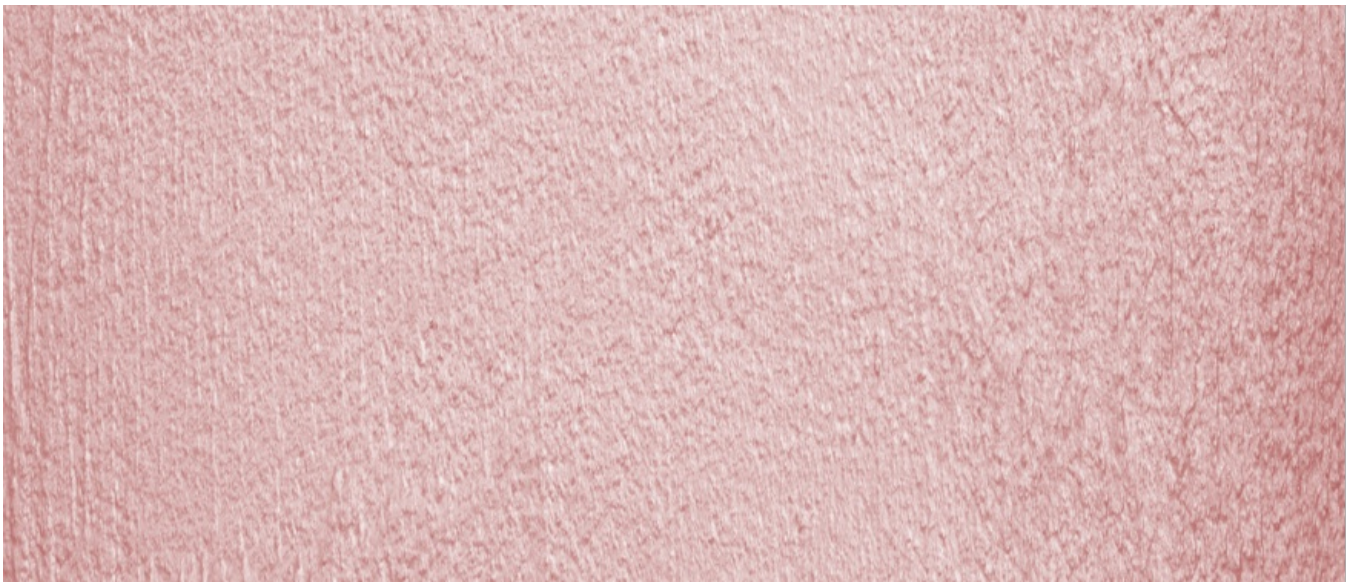

Table of Contents

CHAPTER ONE

What Kubernetes is?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, which facilitates both declarative configuration and automation. It is an open-source container orchestration framework designed to simplify the installation, size and maintenance of containerized applications.

In reality, Kubernetes has developed itself as a de facto norm for container orchestration and is a flagship project of the Cloud Native Computing Foundation (CNCF), sponsored by key players such as Google, AWS, Microsoft, IBM, Intel, Cisco and Red Hat. It has a large, fast-growing ecosystem. Kubernet's services, support and tools are widely available.

It is the most powerful open-source framework for distributed delivery, scaling, management and running of software storage containers. This contains all essential elements and improved scalability possibilities as a full container-centric architecture. Because automation remains the core part of this, technical resource management, adding new capabilities for end-users through scaling up assets has become more apparent with this choice.

Kubernetes makes it easy to build and run software in the framework of micro-services. This is done by creating an abstraction layer on top of a host group, so that development teams can deploy their applications and allow Kubernets to manage:

• Controlling resource consumption by application or team

Evenly spreading application load across a host infrastructure

Automatically load balancing requests across different application instances

Monitoring resource consumption

Moving a software instance from one host to another if there is a shortage of resources on the network, or if the host dies

Automatically exploiting additional resources made available when a new host is connected to the cluster.

Easily executing canary deployments and rollbacks

Kubernetes was originally developed and engineered by Google engineers. Google was one of the early pioneers of Linux container software and has spoken extensively about how everything operates in Google containers. (This is the infrastructure underlying Google's cloud services.)

Google produces more than 2 billion container rollouts a week— all driven by an internal platform; Borg. Borg is Kubernet's predecessor, and the lessons learned from Borg's creation over the years have become the primary influence behind much of Kubernet's innovation.

Kubernetes is an open source project that has become one of the most popular container orchestration tools in the world; it allows you to deploy and manage multi-container applications on a scale. While in reality Kubernetes is most often used with Docker, the most common containerization framework, it can also operate with any container project that conforms to the Open Container Initiative (OCI) specifications for container image formats and runtime. And because Kubernetes is an open source, with very little limitations on how it can be implemented, it can be openly used by anyone who wants to run containers, many of them anywhere they want to run— on-site, in the public cloud, or both.

The main objective of Kubernetes is to mask the difficulty of maintaining a fleet of containers by supplying the REST APIs for the necessary features.

Kubernetes is portable in nature, which means that it can run on various public or private cloud platforms, such as AWS, Azure, OpenStack, or Apache Mesos. It can run on bare metal machines as well.

What Kubernetes Is Not

Kubernetes is not a traditional, all-inclusive PaaS (Service Platform) system. Because Kubernetes runs at the container level rather than at the hardware level, it offers certain generally applicable functionality similar with PaaS services, such as installation, scaling, load balancing, reporting, and tracking. Kubernetes is not monolithic, however, and these default solutions are optional and pluggable.

Kubernetes provides building blocks for developing platforms, but retains user choice and flexibility where it is important.

Kubernetes is not known to do any of the following:

• Does not limit the types of applications supported.

Kubernetes strives to serve a wide range of workloads, including stateless, stateful and data-processing workloads. If an application can run in a container, it should run fine on Kubernetes.

• Does not upload the source code and does not construct the software:

Continuous integration, distribution and configuration (CI / CD) workflows are dictated by corporate environments and priorities as well as by technical requirements.

• Does not provide application-level utilities such as middleware (e.g. message buses), data-processing structures (e.g. Spark), servers (e.g. mysql), indexes, or cloud storage systems (e.g. Ceph) as built-in products. Such components can run on Kubernets and/or can be accessed by applications running on Kubernets through portable mechanisms, such as the Open Service Broker.

• Does not require logging, monitoring or alerting solutions.

It provides some integrations as proof of concept and mechanisms for collecting and exporting metrics.

• Does not provide or allow a setup language / system (e.g. jsonnet).

It provides a declarative API that may be targeted at arbitrary forms of declarative specifications.

• Does not provide or implement systematic computer design, repair, control or self-healing schemes.

Kubernetes is not a pure orchestration device. It completely removes the need for orchestration. The technical definition of orchestration is the execution of a defined workflow: first do A, then do B, then do C. On the other side, Kubernetes consists of a series of separate, composable control processes that continuously move the current state to the desired state. It's not supposed to matter how you get from A to C.

There is also no need for centralized control. This results in a process that is easier to operate and more strong, stable, flexible and extensible.

The Significance of Kubernets

As more and more organizations move to microservice and cloud native architectures that make use of containers, they are looking for strong, proven platforms. Practitioners move to Kubernetes for four main reasons:

1. Kubernetes are making you move faster. In fact, Kubernetes allows you to deliver a self-service Platform-as - a-Service (PaaS) that creates a hardware layer abstraction for development teams. The development teams will order the support they need quickly and efficiently. If they need more resources to handle additional loads, they can get them just as quickly, because all the resources come from infrastructure shared across all your teams.

No more filling out paperwork to request for new machines to operate the query. Just take advantage of the tools developed around Kubernets for automating packaging, deployment and testing, such as Helm (more below).

1. Kubernetes are cost-effective. Kubernets and containers provide for better resource usage than hypervisors and VMs do; since containers are so lightweight, they need fewer CPU and storage resources to run.

2. Kubernetes is an agnostic network. Kubernetes operates on Amazon Web Services (AWS), Microsoft Azure, and the Google Cloud Platform (GCP) and you can also run it on-site. You can switch workloads without needing to reinvent the systems or fully reconsider the infrastructure — which allows you to standardize on a system to escape vendor lock-in.

In fact, companies like Kublr, Cloud Foundry, and Rancher provide tooling to help you deploy and manage your Kubernetes cluster on-premise or on whatever cloud provider you want.

1. Cloud providers will manage Kubernetes for you. As noted earlier, Kubernetes is actually the clear standard for container orchestration devices. It should come as no surprise, though, that big cloud providers are providing plenty of Kubernetes-as - a-Service-offerings. Amazon EKS, Google Cloud Kubernetes Engine, Azure Kubernetes Service (AKS), Red Hat Openshift, and IBM Cloud Kubernetes Service all provide a full Kubernetes platform management, so you can focus on what matters most to you— shipping applications that delight your users.

Why do you need Kubernetes and what can it do ?

Containers are a good way to bundle and run your applications. In a production environment, you need to handle the containers that operate the software to insure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be better if the program were to manage this behavior?

This is how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems in a resilient manner. It will take care of scaling and failover for your software, have delivery instructions, and more. For example, Kubernetes will easily manage the canary implementation of your device.

Kubernetes provides you with:

• Service discovery and load balancing Kubernetes may display a container using a DNS name or using their own IP address. If the traffic to the container is high, Kubernetes will be able to load the balance and distribute the network traffic so that the deployment is stable.

Cloud Orchestration

Kubernetes enables you to dynamically install a storage system of your choice, such as local storage, public cloud services, and more.

Automated rollouts and rollbacks

You can describe the desired state of use of Kubernets for your deployed containers and change the actual state to the desired state at a controlled rate. For example, you can automate Kubernets to create new containers for your deployment, remove existing containers, and transfer all their assets to the new container.

Automatic bin packing: You supply Kubernets with a cluster of nodes that can be used to operate containerized tasks. You inform Kubernets how much CPU and storage (RAM) every container requires. Kubernets will attach containers to your nodes and make the best use of your assets.

Self-healing

Kubernets restarts containers that malfunction, restores containers, destroys containers that do not respond to a user-defined health check, and does not advertise them to clients until they are ready to serve.

Password and configuration management Kubernets lets you store and maintain sensitive information, such as passwords, OAuth codes, and ssh keys. You can deploy and update your secrets and application configuration without rebuilding your container images, and without exposing the secrets in your stack configuration.

How Does Kubernetes Compare To Other Tools?

Container orchestration is a well-deserved popular trend in cloud computing. At the outset, the industry focused on pushing the adoption of containers, moving forward next to deploying containers to scale production. There are a lot of useful resources in this field. To learn about some of the other devices in this room, we'll examine a few of them by comparing their features to the Kubernets.

The key players were Apache Mesos / DCOS, Amazon's ECS, and Docker's Swarm Mode. Each of them has its own position and special capabilities.

DCOS (or DataCenter OS) is similar in many ways to Kubernetes. DCOS pools compute resources into a single task pool. The big difference is that DCOS targets many different types of workload, including but not limited to containerized applications. This renders DCOS appealing to companies that do not use containers for all their software. DCOS also provides a kind of package manager for quick installation of applications such as Kafka or Spark. You can even operate Kubernetes on DCOS because of its simplicity for different types of workloads.

ECS is the entry of AWS into container orchestration. ECS lets you create pools of EC2 instances and use API calls to orchestrate containers across them. It can only be used within AWS and is less detailed than the open source equivalents. It may be of use to people deep within the AWS ecosystem.

Docker's Swarm Mode is Docker Inc's default orchestration tool. Swarm Mode creates a cluster from a variety of hosts in Docker. It has similar features in relation to Kubernetes or DCOS, with one significant difference. Swarm Mode is the only method used to control the dock natively. It ensures that related tools like docker-compose target clusters in Swarm Mode without any modifications.

Basic Recommendations for Usage:

• Using Kubernetes if you only deal with containerized systems that may or may not be just Docker.

If you have a combination of container and non-container programs, using DCOS.

• If you enjoy AWS products and first-party integration, use ECS.

Using Docker Swarm if you want a first-party implementation or a full connection with the Docker Toolchain.

CHAPTER TWO

Kubernetes Architecture

Kubernetes observes the Architecture of the client server. Architecture allows use of different concepts and abstractions. Some of these are variations of existing, familiar notions, but others are specific to Kubernetes.

You may have a multi-master configuration (for high availability), but by definition there is a single master database which serves as a control node

and point of contact. The master server consists of a variety of components, including a kube-apiserver, a kube-controller-manager, a cloud-controller-manager, a kube-scheduler, and a DNS server for Kubernet services. The node components are kubelet and kube-proxy on top of Docker.

Master Components

Below are the main components found on the master node:

•ETCD cluster - It is a simple, distributed key value storage that is used to store Kubernet cluster data (such as number of pods, their status, namespace, etc), API objects, and service discovery details. It can only be used from the API database for security reasons. etcd allows cluster updates for configuration changes with the aid of watchers. Notifications are API queries for each cluster node, etc., that will cause an updating of the information stored in the database.

• Kube-apiserver: Kubernetes API server is a central management entity that receives all REST requests for changes (pods, services, replication sets / controllers and others) to serve as a cluster frontend. Often, this is the only element which connects with the etcd cluster, guarantees that the data is stored in etcd and deals with the network specifics of the pods being deployed.

• Kube-controller-manager-runs a number of separate controller processes in the background (for example, the replication controller controls the number of replicas in the pod, the endpoint controller populates endpoint objects such as services and pods, and others) to regulate the cluster's shared state and perform routine tasks. When there is a shift in the configuration of the system (for example, removing the picture from which the pods are operating, or modifying the parameters in the configuration yaml file), the controller will see the change and start working towards the new desired state.

Cloud-controller manager- It is responsible for managing controller processes that rely on the underlying cloud provider (if applicable). Of example, when the controller needs to check if the node has been terminated or paths, load balancers or volumes in the cloud infrastructure have been set up, all that is done by the cloud-controller-manager.

• Kube-scheduler - It helps to assign pods (a co-located community of containers within which our system processes are running) to various nodes dependent on resource utilization. It reads the operational requirements of the service and schedules it to the most appropriate node. Of eg, if the software requires 1 GB of memory and 2 CPU cores, the pods for that request will be programmed to a node with at least such assets. The scheduler runs every time you need to schedule pods. The scheduler will know the total resources available as well as the resources allocated to the current workloads of each node.

Node (worker) components

Below are the main components contained on the (worker) node:

• Kubelet-main system on the network, periodically adding new or updated pod requirements (primarily through the kube-apiserver) and ensuring that the pods and their containers are stable and operating in the desired state. This component also reports to the master about the health of the host where it is running.

• Kube-proxy-a proxy system that operates on every worker's node to communicate with individual host subnets and reveal resources to the outside world. It forwards the request to the correct pods / containers across the various isolated cluster networks.

How Kubernetes functions

A cluster is the central component of Kubernetes. A cluster consists of many digital or physical computers, each of which performs a specific purpose, either as a master or as a node. Each node hosts groups of one or more containers (which contain your applications) and the master communicates with nodes when to create or destroy containers. At the same time, it tells nodes how to reroute traffic based on new container alignments.

CHAPTER THREE

How To Build Kubernetes

Creating Kubernets is simple if you take advantage of the built-in container environment. A book session will help guide you through the comprehension of this building process.

Specifications

• Docker (using Mac OS)

You can either use Docker for Windows or Docker for IOS. Note: You will want to set the Docker VM to have at least 4.5 GB of initial memory or it will probably fail to build.

• Linux with local Docker Download Docker according to the OS instructions.

• Virtual Docker Engine Use a huge computer in the cloud to create quicker. It's a bit trickier, but look at the chapter first.

• Available Google Cloud SDK You will download and customize Google Cloud SDK if you want to submit the update to Google Cloud Storage and you can easily remove it otherwise.

While it is possible to build Kubernets using a local golang setup, there is a building process running in a Docker tank. This simplifies the initial setup and ensures a very consistent build and test environment.

Core Scripts

The following scripts can be located in the build / directory. Note that all scripts must be run from the Kubernet root directory.

build/run.sh : Run a program in a dock build tank. Popular invocations:

build/run.sh make : build linux binaries in the tank. Pass choices and packages as needed.

Build / run.sh make cross : build all binaries for all platforms

build/run.sh make kubectl KUBE BUILD PLATFORMS= darwin / amd64 : Build a specific binary for a specific platform (in this example, kubectl and darwin / amd64)

build / run.sh make test : run all unit tests

build / run.sh make test-integration : run integration test

build / run.sh make test-cmd : run CLI tests

build / copy-output.sh: It will also copy unique file patterns that are created as part of the build method. This will run automatically as part of build / run.sh.

Build/make-clean.sh: Clean the contents of output, delete any locally created file images, and disable the software container.

Build / shell.sh: Drop to a bash shell in a build container with a snapshot of the current repo code.

Basic Flow

The scripts that are directly under build / are used to build and test. They will make sure that the kube-build Docker image is built (based on build / build-image / Dockerfile) and then execute the appropriate command in that container. Both of these scripts guarantee that the right data is stored from run to run for gradual builds and that the tests are copied out from the box.

The kube-build container image is created by first creating a "context" directory in output/images/build-image. It's done there instead of at the root of the Kubernetes api to minimize the amount of information that we need to load while creating an object.

There are three separate instances of the container running from this image. The first is a "data" container to store all the data that needs to be maintained across to support incremental builds. First, there is a "rsync" module that is used to transfer data to and from the software server. Finally, there is a "build" container that is used to actually do build actions. The data container continues to run while the "rsync" and build containers are deleted after each use.

"rsync" is used transparently behind the scenes to move data in and out of the container effectively. This is going to use the ephemeral port selected by Docker. You can adjust this by defining the KUBE RSYNC PORT env variable.

All Docker names are suffixed with a hash extracted from the directory path (to make simultaneous use on items like CI machines) and a version number. When the version number increases, the whole state is cleared and the clean construct is begun. It helps the building environment to be updated and communicates to CI networks that old objects need to be discarded.

Proxy Settings

If you are behind a proxy and require these scripts to use the docker-machine to set up your local VM for you on macOS, you need to export proxy settings to create Kubernets, the following environment variables should be specified.

Export

KUBERNETES HTTP PROXY= http:/username: password@proxyaddr: proxyport

Optionally, you can define non-proxy address for Kubernetes build, e.g. send KUBERNETES NO PROXY=127.0.0.1 If you are using sudo to create Kubernetes construct for instance make quick-release, you need to run sudo-E make quick-release to transfer

Really Remote Docker Engine

You can use a Docker Engine running remotely (under your desktop or in the cloud). Docker must be configured to link to that computer and forward the local rsync port (via SSH or nc) from the localhost to the remote machine.

To do this effectively with GCE and a docker server, do something like this: #Build a remote docker system on GCE. This is a pretty beefy machine with SSD Disk.

KUBE*BUILD*VM= k8s

Build*KUBE*BUILD*GCE*PROJECT= docker-machinecreate\—driver= google \—google-project=${KUBE BUILD GCE PROJECT} \—google-zone= us-west1-a \—

google-machine-type= n1-standard-8 \—

google-disk-size=50 \—

google-disk-type= pd-sd \${KUBE BUILD VM} #Set local docker to talk to that machine $eval(docker-machine env$UBUB

Releasing

The create / release.sh script is going to build an update. It will build files, run trials, and (optionally) create runtime docker photos.

The primary source is tar: kubernetes.tar.gz.

This includes;

• Cross-compiled system services used.

• Script (kubectl) for selecting and operating the right platform based binary user.

• Examples

• Server configuration scripts to various clouds

• Tar file comprising all database binaries

In addition, some other tar files were created:

kubernets-client-*.tar.gz Client binaries for a specific platform.

• kubernets-server-*.tar.gz Server binaries for a specific platform.

When building final release tars, they are first staged in output/release-stage before being tar'd up and put in output/release-tars.

Reproducibility makes release, its variant makes quick release, and Bazel all provide a hermetic build environment that should provide some level of reproducibility for builds. It's not hermetic to make yourself.

The Kubernets development environment supports the SOURCE DATE EPOCH environment variable defined by the Reproducible Builds program, which can be set to the UNIX epoch timestamp. This will be used to create timestamps embedded in compiled Go binaries, and maybe someday Docker images as well.

A fair configuration for this parameter is to use the commit timestamp from the tip of the tree being built; this is what the Kubernetes CI system uses. For example, you can use the following one-liner.

CHAPTER FOUR

Concepts And Design Principles

Kubernetes Concepts

Using Kubernetes requires understanding the different abstractions that it uses to represent the state of the system, such as services, pods, volumes, namespaces, and deployments.

Pod–In general, it applies to one or more containers that should be managed as a single application. A pod encapsulates device modules, computing space, a special network Identifier, and other settings for operating containers.

Service-The pods are volatile, i.e. Kubernetes does not guarantee that a given physical pod will be kept alive (for example, the replication controller might kill and start a new set of pods). Instead, the service represents a logical set of pods and acts as a gateway, allowing (client) pods to send requests to the service without having to keep track of which physical pods actually make up the service.

Volume-similar to the volume of the container in Docker, but the size of the Kubernets extends to the whole system and is placed on all the containers in the row. Kubernetes guarantees that data is stored throughout the restart of the container. The pressure will only be replaced when the pod is broken. Also, a pod can be correlated with several volumes (possibly of different types).

Namespace-a virtual cluster (a single physical cluster can run multiple virtual clusters) designed for environments with multiple users spread across multiple teams or projects to address concerns. Resources within a namespace must be unique and can not access resources in another namespace. In addition, a namespace resource quota may be allocated to avoid consuming more than its share of the total resources of the physical cluster. Namespaces allow you to create virtual clusters at the top of a physical cluster. Namespaces are designed for use in systems with multiple users distributed through multiple teams or programs. They delegate capacity limits and theoretically separate cluster resources.

_____

Deployment-specifies the required state of the pod or replica set in the yaml file. The deployment controller will then gradually update the environment (such as creating or deleting replicas) until the current state matches the desired state specified in the deployment file. For example, if the yaml file defines 2 replicas for a pod but only one is currently running, an extra one will be created. Note that replicas managed through deployment should not be manipulated directly, only through new deployments.

Deployments and replicas Deployments is a YAML object that defines the pods and the number of container instances, called replicas, for each pod. You define the number of replicas that you want to run in the cluster using the ReplicaSet that is part of the deployment object. Therefore, for instance, if a pod node fails, the replacement array would insure that another pod is prepared for another usable node.

The DaemonSet deploys and operates a particular daemon (in a pod) on the nodes which you choose. Most of the time they are used to provide services or maintenance to pods. A daemon set, for example, is how the New Relic Infrastructure agent is deployed across all cluster nodes.

_____

Labels Labels are key / value pairs that can be assigned to pods and other objects in Kubernets. Labels allow Kubernete operators to arrange and pick subsets of objects. For example, when monitoring Kubernet objects, labels let you quickly drill down to the information you're most interested in.

Stateful sets and persistent storage volumes StatefulSets give you the ability to assign unique pod IDs if you need to move pods to other nodes, maintain networking between pods, or maintain data between them. Similarly, persistent storage volumes provide storage resources for a cluster to which pods may request access as they are deployed.

Some valuable modules These Kubernetes components are helpful but are not needed for standard Kubernetes functionality: Kubernetes DNS Kubernetes offers this framework for DNS-based service discovery between pods. In contrast to any other DNS servers that you may use in your network, this DNS server functions.

Cluster-level logs If you have a logging tool, you can integrate it with Kubernets to extract and store applications and system logs from within a cluster, written to the standard output and to the standard error. If you want to use cluster logs, it is important to note that Kubernetes does not provide native log storage; you need to provide your own log storage solution.

Helm: managing Kubernet applications Helm is the application package management registry for Kubernetes maintained by the CNCF. Helm "charts" are pre-configured software application resources that can be downloaded and deployed in your Kubernet environment. According to the 2018 CNCF report, 68% of respondents said that Helm was the chosen package management tool for Kubernet's applications. Helm charts can help DevOps teams speed up running software in Kubernets; they can use established charts that they can exchange, update, and distribute in their Dev and development environments.

Kubernetes and Istio: popular pairing In the architecture of microservices such as those running in Kubernetes, a service mesh is an infrastructure layer that allows your service instances to communicate with each other. The service mesh also lets you configure how critical actions such as service discovery, load balancing, data encryption, and authentication and authorization are performed by your service instances. Istio is one such service mesh, and the current thinking of tech leaders, like Google and IBM, suggests that they are becoming increasingly inseparable.

For example, the IBM Cloud team uses Istio to address the issues of control, visibility, and security that it has encountered while deploying Kubernets on a massive scale. Specifically, Istio helps IBM: connect services together and control traffic flow Secure interactions between

microservices with flexible authorization and authentication policies Provide a control point so that IBM can manage production services Observe what happens in their services via an adapter that sends Istio data to New Relic — allowing IBM to monitor microservice performance.

Kubernetes Design Principles Kubernetes was designed to support the features required by highly available distributed systems, such as (auto)scaling, high availability, security and portability.

Scalability-Kubernetes ensures horizontal scaling of pods on the basis of CPU utilization. The limit for CPU usage is configurable and Kubernetes can automatically start new pods when the threshold is reached. For eg, if the limit is 70 per cent for CPU, but the project is actually growing to 220 per cent, so gradually 3 more pods will be installed so that the total CPU utilization is below 70 per cent. If several pods are required for a specific application, Kubernetes offers the load balancing functionality across them. Kubernetes also supports the horizontal scaling of state-of - the-art pods, including NoSQL and RDBMS databases, through Stateful sets. A Stateful set is a similar concept to a Deployment, but ensures that storage is persistent and stable even when the pod is removed.

High Availability-Kubernetes tackles high availability at both software and service stage. Replica sets ensure that the (minimum) number of replicas of the stateless pod for a given application is running. Stateful sets play the same role as stateful packs. At the network stage, Kubernetes supports multiple distributed storage backends such as AWS EBS, Azure Cloud, Google Persistent Disk, NFS, and more. Adding a reliable, available storage layer to Kubernet ensures high availability of state-of - the-art workloads. In addition, each of the master components can be configured for multi-node replication (multi-master) to ensure higher availability.

Security-Kubernet addresses security at multiple levels: cluster, application and network. The API endpoints are secured by means of the Transport Layer Security (TLS). Just authorized users (either network accounts or regular users) may run cluster operations (via API requests). At the system stage, Kubernet secrets will store sensitive information (such as passwords or tokens) per cluster (virtual cluster if namespaces are used, otherwise physical). Remember that secrets can be obtained from any pod in the same group. Network policies for accessing pods can be defined in the deployment. The network policy defines how pods will interact with each other and with other network endpoints.

Portability-The Kubernet portability manifests in terms of operating system options (a cluster can run on any mainstream Linux distribution), processor architectures (either virtual machines or bare metal), cloud providers (AWS, Azure or Google Cloud Platform), and new container runtimes, besides Docker, can also be added. The federation concept can also support workloads across hybrid (private and public cloud) or multi-cloud environments. This also facilitates the provision of fault zone sensitivity within a single cloud provider.

_____

Kubernetes clusters The highest-level Kubernetes concept, the cluster, applies to the collection of machines running Kubernetes (the clustered system itself) and the containers it handles. The Kubernet cluster must have a master, a system that controls and controls all other Kubernet machines in the cluster. A highly available Kubernet cluster replicates the master's facilities across multiple machines. But the task scheduler and controller-manager work just one boss at a time.

Kubernetes nodes and pods Each cluster contains nodes of Kubernetes. Nodes could be physical machines or VMs. Again, the idea is abstraction: whatever the app is running, Kubernetes handles the deployment on that substrate. In fact, Kubernetes makes it possible to ensure that certain containers run only on VMs or on bare metal.

Nodes run pods, the most simple Kubernet entities that can be generated or controlled. That pod is a single instance of an application or a running system in Kubernet and consists of one or more containers. Kubernetes begins, ends, then replicates all of the containers in the pod as a group. Pods hold the user's focus to the program, rather than to the containers themselves. Details on how Kubernets needs to be configured, from pod status up, are stored in Etcd, a distributed key-value store.

Pods are created and destroyed on nodes as needed to comply with the desired state specified by the user in the pod definition. Kubernetes provides an abstraction called a controller to deal with the logistics of how pods are spun up, rolled out, and spun down. Controllers come in a variety different flavors depending on the type of software being handled. For example, the recently introduced "StatefulSet" controller is used to manage applications that need a permanent state. Another type of controller, deployment, is used to scale the app up or down, update the app to a new version, or roll back the app to a known-good version if there's a problem.

Kubernetes as a service Nowadays, there are so many services that Kubernetes gives you a free service. It's not about homemade marmalade, it's about technology. Just purchase it from a better player, if you can. Facebook, Amazon, Microsoft, IBM... all of them have a Kubernet system and all operate with the same pricing model. A lot of vendors give you a free tier or credit trial period so it's easy to start playing with. For example, Azure has a number of features during source code integration, but most of the important features apply to all of them.

Another great concept of Kubernets as a platform is that it doesn't allow the software to be connected to the internet. Since your application is a raw container and this is orchestrated by some Kubernet-compliant configuration files, you can move to azure to google and vice versa with little effort (at least without changing the source code). Therefore, the Kubernetes service is free and you only pay for the equipment, where "hardware" represents a virtual machine used by Kubernetes.

Kubernetes Architecture has the following main components:

• Master nodes

• Worker / Slave nodes

• Distributed key-value store(etcd.)

Master node This is the entry point for all administrative tasks that are responsible for managing the Kubernetes cluster. There may be more than one master node in the cluster to check for fault tolerance. More than one master node places the device in a High Availability state, one of which will be the key node which executes all the tasks. To manage the cluster state, use etcd, in which all master nodes connect to it.

Master node components are as follows

• API database: executes all administrative tasks through the master node API service. The REST commands are sent to the API database which validates and handles applications. The resulting state of the cluster is stored in the distributed key-value store after the request.

• Scheduler: The scheduler sets the tasks for the slave nodes. This records data on the use of assets for each slave node. The work is scheduled in the form of Pods and Services. Before scheduling the task, the scheduler also takes into account the quality of the service requirements, the data locality, affinity, anti-affinity, etc.

• Controller manager: also known as controllers, it is a daemon that regulates the Kubernetes cluster that manages the various non-terminating control loops. It also performs life-cycle functions such as namespace creation and life-cycle, event garbage collection, finished-pod garbage collection, cascading-deletion garbage collection, node garbage collection, etc.

Basically, the controller monitors the desired state of the objects it manages and monitors their current state through the API server. If the current state of the entities it controls does not match the target state, the control loop may take corrective action to ensure that the current state is the same as the desired state.

• ETCD

This is a decentralized key-value store that stores the state of the cluster. It can be part of the Kubernet Server, or it can be installed externally. Etcd is written in the programming language of Go. In Kubernetes, besides storing the cluster state (based on the Raft Consensus Algorithm), configuration information such as subnets, ConfigMaps, Secrets, etc. are also kept.

A raft is a consensus algorithm designed to be an alternative to Paxos. Consensus issues involve multiple servers deciding on values; a common problem that occurs in the sense of replicated state machines. Raft defines three different roles (Leader, Follower, and Candidate) and reaches consensus through an elected leader, Worker Node (formerly minions) Worker Node-Kubernetes Architecture-Edurekalt is a physical server or you can say a VM running applications using Pods (a pod scheduling unit) which is controlled by a master node. Pods are stored on a physical server (worker / slave node). We link to nodes to access software from the outside world.

The main components are as follows; Server runtime: we require a server runtime on the worker node to operate and handle the lifecycle of the server. Docker is sometimes referred to as a container runtime, but to be specific, Docker is a framework that utilizes containers as a database runtime.

Kubelet : is an agent who communicates with the Master node and executes nodes or worker nodes. It gets the Pod parameters through the API database and runs the Pod-associated containers and maintains that the Pod-associated containers are working and stable.

Kube-proxy: Kube-proxy operates on each node to communicate with specific host subnets and to insure that services are available to external parties. This acts as a network proxy and a load balancing tool for a system on a single worker node and handles network forwarding of TCP and UDP packets.

It is a network proxy that runs on each node of the worker and listens to the API server for each creation / deletion of the service endpoint. For each endpoint of the Network, kube-proxy lays out the routes so that they can access it.

Setting Up The Cluster

Now that we've addressed the individual components which operate on each of the devices in the cluster, we should think about how you're really distributing them across a number of machines. Based on your use case, you may want to set up the Kubernetes framework based on any of the following topologies.

Single Control-Plane Cluster: This is an appropriate option when you are just beginning to test Kubernets for your applications. In this cluster type, the master or control-plane components run only one node in your cluster. It makes it easy to install and maintain it at the cost of reliability and flexibility choices. You can choose from the following options when it comes to your worker nodes: Single Node Cluster With this topology, all

master and worker node components operate on a single node. Thus, a single machine will fulfill all the roles of control-planet, etc., and worker. Referring to this as a cluster is a bit of a stretch, as it consists only of a single machine, but we're going to use that language to remain consistent.

This type of cluster is really only suitable for development work, running a few tests or CI environments, and should not be used in production. There are a variety of products / tools such as K3s, RKE, Minikube, and Kubeadm that you can use to bring the cluster up and running within minutes.

Single Manager, Several Worker Nodes This topology helps to distribute the functions by splitting the worker node components from the manager nodes. So all master components, such as the kube-apiserver, etc., the kube-scheduler, and the controller-manager, will run as before on a single master node. Nonetheless, one or more dedicated nodes will be used to operate the components of the job.

Single node cluster This topology is more fault-tolerant than a single node cluster, but is still not recommended for development. It can manage broader workloads, but it is still not fault-tolerant when it comes to the kube-apiserver or the like, without which the cluster can not run.

HA Kubernet Cluster is known for its resilience and reliability. This can be achieved by ensuring that the cluster does not have a single point of failure. Because of this, in order to have a highly available cluster, you need to have multiple master nodes. All master modules, such as the kube-apiserver and so on, can be scaled up to ensure high availability. Each of the replicas of the etcd copy the information contained for the cluster. This data redundancy ensures high reliability and availability. It is also recommended that master nodes be spread across different regions or zones to manage zone failures.

Etcd Quorum

Etcd stores the specification and state of the Kubernet cluster. You need to have multiple nodes running etcd to make your HA cluster. Etcd requires most of the nodes, considered a quorum, to decide on cluster changes. In the case of a cluster of n representatives, the quorum is estimated as (n/2)+1 (one greater than 50%).

So to make your Kubernetes cluster tolerant to partial failure, etc., you will need to run on at least three nodes. In a cluster of 3 etcd nodes, the quorum will be 2, so it can still function if one of the nodes goes down. You can choose from the following topologies for the HA cluster depending on how you set up etcd nodes:

Master Nodes with Co-Located Control-Plane and ETCD A single control-plane master is scaled horizontally in this cluster type, so you have multiple master nodes. Each of them runs all the master components, including the control-plane and so on. The kube-apiserver interacts with the local etcd member on each node. You will need at least 3 master nodes and one or more dedicated worker nodes to bring up the HA cluster with this topology. This is also defined as the stacked etcd topology: stacked etcd cluster External Etcd cluster In this topology, dedicated etcd nodes run separately from the nodes operating the other control-plane elements. Topology with co-located control plane and components etc. faces the risk of failure of the coupling. If a master node goes down, it is equivalent to losing a control-place node along with a node, etc. This can be avoided by decoupling the control-plane and nodes, etc.

In this topology, every control-plane node runs kube-apiserver, kube-scheduler, and kube-controller-manager. The Etcd members are running on separate nodes. Growing user, etc., may interact with the apiserver on each of the control-plane nodes. With this model, even if the control plane node goes down, the nodes are not affected, and vice versa. This way, data redundancy is not affected as much as it would be in the first HA topology. The only downside to this model is that this configuration needs twice as many nodes as normal.

CHAPTER FIVE

How To Deploy And Manage Applications On Kubernetes

In this chapter of the book, we will show how to deploy an application to a Kubernetes cluster. They will use a basic example container to walk through how to build a Deployment, how to upgrade the running framework with kubectl, and how to scale the application out by launching further container instances within the same Deployment.

Deploying an Application

Within Kubernetes Kubernetes is an open source project designed specifically for container orchestration. Kubernetes offers a number of key features, including multiple storage Interfaces, container health checks, manual or automated scaling, rolling updates and system discovery. Apps can be downloaded to a Kubernetes cluster via Helm maps, which provide simplified package management functions.

If you are new to Kubernetes and Helm charts, one of the easiest ways to discover their capabilities is with Bitnami. Bitnami offers a number of secure, production-ready Helm maps to install popular software apps, such as WordPress, Magento, Redmine and many more, in a Kubernetes cluster. And, if you 're creating a custom program, it's even possible to use Bitnami's Helm maps to prepare and launch it for Kubernetes.

This tutorial takes you through the cycle of bootstrapping an instance MongoDB, Express, Angular and Node.js (MEAN) framework on a Kubernetes cluster. It uses a custom Helm chart to create a Node.js and MongoDB environment and then clone and deploy a MEAN application

from a public Github repository into that environment. Once the application is deployed and working, it also explores some of Kubernetes' most interesting features: cluster scaling, load-balancing, and rolling updates.

Assumptions And Prerequisites

This guide focuses on deploying an example MEAN application in a Kubernetes cluster running on either Google Container Engine (GKE) or Minikube. The example application is a single-page Node.js and Mongo-DB to - do application available on Github.

The tutorial incorporates the following assumptions:

You have a Kubernetes 1.5.0 (or later) cluster.

You have kubectl downloaded and setup to operate with your Kubernetes cluster.

You've also downloaded and setup git.

You've got a basic understanding of how containers work. Read more about the Wikipedia and ZDNet bins.

TIP: If you don't already have a Kubernet cluster, the easiest way to get one is via GKE or Minikube. Please refer to our Starter Tutorial for detailed instructions.

NOTE:

GKE is preferred for enterprise implementations because it is a production-ready system with assured uptime, load balancing and container networking functionality. That said, the commands shown in this document can be used for both GKE and Minikube. Commands specific to one or the other platform are explicitly referred to as such.

Step 1: Validate The Kubernets Cluster First, make sure that you are able to connect to the kubectl cluster data. This order is also a good way to get your cluster's IP address.

Kubectl cluster-info You should see output similar to the following: cluster information It's also a good time to get some information about the physical nodes in the kubectl cluster: kubectl get nodes Sample output is shown below: cluster information TIP: for detailed cluster health and status, visit the Kubernetes dashboard.

Phase 2: Download Helm And Tiller To download Helm, execute the following commands: curl https:/raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get helm.sh chmod 700 get helm.sh./get helm.sh Once the installation process is complete, download Helm's server-side equivalent, Tiller, with helm init command: helm init You should see anything close to the performance below: Tiller deployment The pod consists of one or more containers that can communicate and share data with each other. Pods make it easy to scale applications: scale by adding more pods, scale by removing pods. Find out more about the pods.

The Helm Map used uses the example to - do framework as two pods: one for Node.js and the other for MongoDB. This is considered a best practice because it allows for a clear separation of concerns and also allows pods to be scaled independently (see this in the next section).

NOTE: The Helm Chart used in this guide has been developed to show the capabilities of both Kubernetes and Helm and has been tested to work with an example to - do application. It can be modified to function with other MEAN systems, but some modifications may be required to link the MongoDB module to the database pod.

To deploy the sample application using the Helm Chart, follow the following steps: Clone the Helm Chart from the Bitnami Github repository: git clone https:/github.com/bitnami/charts.git cd charts / incubator / mean Check and install missing helm dep dependencies. The Helm Map used in this instance relies on the MongoDB map in the official database, so the commands below will be used to define and download the required dependency.

Helm Dep List Helm Dep Update Helm Dep Build Here's what you should see: Dependency installation with Helm Lint The Helm Lint Chart to ensure that there are no errors.

Helm lint, man.

Attach the Helm Map with the helm mounted. This will generate two pods (one for the Node.js system and one for the MongoDB application). Pay special attention to the output NOTES section, as it contains important information for accessing the application.

_____

NOTE: If you do not know the title of the update with the* download helm. —name my-todo-app— set serviceType= LoadBalancer You should see something like the output below when the chart is installed.

Application Deployment with Helm Unlike cloud platforms, Minikube does not support a load balancing device, so if you are deploying the application on Minikube, use the following command instead: helm install. —name my-todo-app— set serviceType= NodePort

You should see the results below as the map is mounted on the Minicube.

Application deployment with Helm

Get the URL for the Node application by executing commands shown in the output of the install helm or by using the helm status of the my-todo-app and checking the output for the external IP address.

If you have deployed the application to GKE, use these commands to obtain the URL for the Node application: export SERVICE IP=$(kubectl get svc— namespace default my-todo-app-mean-o jsonpath='{.status.loadBalancer.ingress[0].ip}') http:/$SERVICE IP / If you have deployed the application to Minikube, use these commands instead of obtaining the URL for the Node application: export NODE PORT=$(kubectl ge Here's what it should look like: Cluster-running application To debug and diagnose deployment problems, use kubectl to get pods-l app= my-todo-app-mean. When you stated a different release name (or did not specify one), continue to use the actual release name from your delivery.

To uninstall and reinstall the Helm Map at any time, use the Uninstall button shown below. The additional— purge method eliminates the launch title from the store so that it can be reused later.

Helm delete — purge my-todo-app

Phase 4: Discover Kubernets And Helm TIP: When the software is working on Kubernets, read our guide to conducting more complicated post-deployment activities, like setting up TLS with Let's Encrypt Certificates and executing rolling updates.

Scale Up (or Down) NOTE: For clarity, this paragraph focuses only on the scaling of the Node.js stack.

As more and more users access the software, there is a need to scale up to handle the increased load. Conversely, during periods of low demand, it often makes sense to scale down and maximize the use of capital.

Kubernetes offers the kubectl scale command to scale up or down the amount of pods in the deployment. Learn more about the command on the kubectl scale.

Check the number of pods currently running for each device with the helm status order, as shown below: helm status my-todo-app The display will give you one running instance of each pod.

Code state Then scale Node.js up to three copies using the kubectl scale command below.

Kubectl scale— replicates 3 deployment / my-todo-app — mean Check the status as before to check that you have three Node.js pods.

Server state

Then scale it back to two with the following command: kubectl scale— replicas 2 deployment / my-todo-app — mean Check the status as before to test that you have two Node.js pods.

Configuration state

The key feature of Kubernetes is that it is a self-healing system: if one or more pods in the Kubernetes cluster come to an abrupt end, the cluster would immediately spin replacements. This ensures that the required number of pods is always running at any given time.

Use the kubectl get pods command to see this in action to get a list of running pods, as shown below:

Server Overview

As you can see, this node has been scaled up to 2 clusters in Node.js. Next, choose one of the Node.js pods and simulate the fall of the pod by deleting it with a command like the one below. Replace the POD-ID location owner with the real pod identifier from the kubectl performance of the pods command.

_____

Kubectl

uninstall pod POD-ID Now run Kubectl get pods-w again and you'll see that Kubernetes immediately replaced the failed pod with the new one: Cluster Self-Healing When you keep watching the performance of Kubectl get pods-w, you'll see the status of the new pod change quickly from "Pending" to "Going." Switch Traffic Among Pods

It's easy enough to turn two (or more) replicas of the same contai. When you deploy an application to the Kubernetes cluster in the cloud, you have the option to automatically create a cloud load balancing device (external to the Kubernetes cluster) to direct traffic between the pods. This load balancing framework is an instance of a Kubernetes Server device. Find out more about the Kubernet products.

When deploying an application to GKE with Helm, the serviceType command used the option to create an external load balancing device, as shown below: helm change. -name my-todo-app— set serviceType= LoadBalancer When activated in this way, Kubernetes will not only create an external load balancing network, but will also be responsible for setting up a load balancing computer with the internal IP address of the pods, setting up firewall regulations, and so on.

For specifics of the load balancing system, use the kubectl summary of the svc control as shown below: kubectl defines the svc my-todo-app Load balancing software overview LoadBalancing Ingress sector, which specifies the IP address of the load balancing unit, and the Endpoints field, which specifies the internal IP address of the three Node.js pods in use.

Similarly, the Port field specifies the port that the load balancer listens to for connections (in this case, 80, the standard Web server port) and the NodePort field specifies the path to the internal cluster node that the pod uses to view the network.

Obviously, this doesn't work the same way for a locally running minicube band. Look back at the Minikube setup and you'll see that the default serviceType has been modified to NodePort. It exposes the network to every node in the cluster on a different socket.

Place your microphone. —name my-todo-app— set serviceType= NodePort Test this by verifying the service specifics with kubectl describes. kubectl describes svc my-todo-app NodePort system overview

The main difference here is that instead of an existing network load balancing service, Kubernetes provides a device that listens to incoming requests on each node and guides them to the fixed open port on each e.

_____

Perform Rolling Updates (And Rollbacks) Continuous updates and rollbacks are important benefits for applying apps to the Kubernets cluster. Through rolling updates, devops teams will execute zero-downtime improvements to the software, which is an important consideration for the production environment. At the same time, Kubernetes also supports rollbacks, which make it easy to get back to the previous iteration of the application without a service outage. Find out more about the roll-up notifications.

Helm makes it easy to update the programs with the helm upgrade button, as shown below: helm upgrade to my-todo-app.

Check the status of the update with the helm history command shown below: helm history my-todo-app Here's what it looks like: NodePort system overview As shown in the display, the software has been updated and now revision #2 is working.

While an upgrade is ongoing, it is necessary to determine the same criteria as when the chart was first deployed. This is especially important when it comes to passwords, because updates will use the same credentials that were installed during the initial deployment. For example, if the original installation is done with the command helm install— name my-app— set app.password=secret, the equivalent update command would be helm uninstall my-app— set app.password=secret NOTE: when updating, always check the map documents to see if there are any improvements between the latest version of the graph and its previous iterations. Broken changes have been made by changing the major and minor version numbers of the chart; patch versions are generally safe to delete.

_____

Rollbacks are just as simple-just use the helm rollback command and set the revision number to roll back to. For example, to go back to the original version of the application (revision #1), use this command: helm rollback my-todo-app 1 When you check the status with the helm history, you will see that revision #2 has been superseded by a copy of revision #1, this time labeled as revision #3.

Node Port Framework Description

By now, you should have a clear understanding of how some of the core features available in Kubernets, such as scaling and automatic load balancing, operate. You should also grasp that Helm diagrams make it easier to implement standard Kubernet activities, including installing, modifying, and rolling back services.

For our instance, we will use the Rancher pre-built box. We will carry out the following operations to add it to the Kubernetes cluster: Create a simple YAML file to install our code inside Kubernetes Modify our setup to link to a new version and roll over the currently active version to modify our deployment Scale the system to create a fully functional application inside Kubernetes Those represent a very quick series of actions that we need.

Defining the Application Deployment

To get started, we need to explain a simple application deployment which Kubernetes can understand by writing a YAML file. YAML is a human-

readable serialization language that Kubernetes can read and translate.

Our YAML document must describe a Deployment object that launches and manages our software container. You should copy the following folder, which we will call testdeploy.yaml to reproduce this example on your own cluster: cat testdeploy.yaml apiVersion: apps / v1 type: Deployment of metadata: name: mysite labels: app: mysite spec: replicas: 1 selector: matchLabels: app: mysite template: metadata: app: mysite spec: containers:- name: m YAML generates a Kubernets Deployment entity.

The deployment specification requests a single replica spawned from a Pod template that launches a Kellygriffin / Hello-based container: v1. The requirement indicates that the container must listen to port 80.

Once the document has been saved, you can add it to your cluster: kubectl apply-f testdeploy.yaml deployment.apps/mysite generated You can check the details of the pod being deployed by typing: kubectl get pods Record the name of your server. Our container was called mysite-59564d59f5-xz9vd for the release that we released, but yours will be special.

Once you have a container title, you can request a web server running on a localhost database by typing: kubectl exec-it < container name > curl localhost You should get an answer that verifies the framework and the version that we used: < html> < head> < title > Hello World This is version 1 of our Application < /html > It ensures that our rollout has been successful and that our application is working correctly.

Now that we have a version running on our Kubernets database, we can monitor and modify it as circumstances dictate. Kubernetes will take care of many distributed management tasks, but there are still occasions when we want to control the actions of our applications.

To demonstrate this, we will update the application version associated with our deployment. Editing the YAML configuration document that we created earlier will not support, as our software is already running within the cluster. Alternatively, we need to change the definition as contained in the actual cluster.

With the Kubectl Edit Control, we may change external artifacts. The purpose of the order is the form of an entity and the title of an object, separated by a forward dash. For our example, we can modify our deployment specification by type: kubectl edit deployment / mysite The deployment specification will be opened in the system's default editor.

Once inside, you need to adjust the following line: change the following line: the original line: image: kellygriffin / hello: v1 Replaced with: image: kellygriffin / hello: v2 If the default editor has been changed to vim, you can save and exit after finishing by hitting the escape key and typing: wq.

When you save the file, Kubernetes will be able to recognize the configuration difference and automatically start updating the Deployment within the cluster.

You can ensure that this process has been completed by checking the pods on your cluster again: cubectl get pods As we've seen before, this will display information about the pods on our cluster, including the names of the containers that are deployed with each pod. Please enter the name of your container again.

You can note that the name of the container is different when you look closely. In our case, our new container is named mysite-5bcfff5d56-n9zvf, but again, yours is going to be different.

We will re-check the web server running on port 80 by telling the container to request localhost: kubectl exec-it < your domain name > curl localhost This time we should see a modified message associated with version 2 of our container that we defined when we revised our Deployment spec: This helps us validate that the actual container within our Deployment has been replaced, based on the new image we specified.

Scaling Applications

Now that we've shown how to patch our applications by changing the Deployment requirements, we can also explore how to scale our software using Kubernet's built-in replication primitives.

With the kubectl scale command, we can modify the scale of our deployment. To complete our application, we need to provide the amount of replicas that we want as well as the Kubernetes entity that we want to reach (in this case, it's our deployment / mysite object).

To scale our Deployment from a single replica to 2, we can type: kubectl scale —replicas=2 deploy/mysite deployment.extensions/mysite scaled We can check on the progress of the scaling operation by asking for the details on our Deployment object: kubectl get deploy mysite.

_____

Clean Up Deployment

We've made a deployment, updated it, and scaled it. Since this is not a real workload of production, we should remove it from our cluster once we're done to clean up ourselves.

We just need to uninstall the Deployment entity and remove the tools we've set up. Kubernetes will automatically remove all other child services linked to it, such as the pods and containers it manages.

Delete Deployment by Typing: delete kubectl by deploying my site deployment. Mysite Extensions Disabled You can double check that the services have been disabled by attempting to view the Deployment specifics or by telling Kubernetes to list the Pods in the default namespace: kubectl get deployed mysite kubectl get pods Such commands will mean that the Deployment and all its related assets have been deleted.

The port flag specifies the port number installed on the Load Balancer, and the — target-port flag specifies the point number that the hello-app container is listening to.

Note: GKE assigns an existing IP address to the Server tool, not to the Deployment. If you want to figure out the existing IP given by GKE for your request, you should search the Network with the kubectl network command: kubectl get network Output: Id CLUSTER-IP Existing-IPPORT(S) AGE hello-web 10.3.251.122 203.0.113.0 80:30877/TCP 3d

After you have defined the current IP address for your software, copy the IP address. Point your client to this URL (such as http:/203.0.113.0) to make sure your request is available.

Scale up the software

You add further replicas to your application's Deployment tool by using the kubectl scale order. To add two more replicas to your Deployment (for a total of three), run the following command: kubectl scale deployment hello-web— replicas=3 You can see the latest replicas operating on your cluster by running the following commands: kubectl get deployment hello-web output: NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE hello-web 3 3 3 2 1 m.

The load balancer you described in the previous step will automatically start the routing of traffic to these new replicas.

Install a new version of your GKE app's rolling upgrade mechanism to ensure that your software remains up and usable, even as the framework replaces instances of your old container picture with a new one across all operating replicas.

You can create an image for the v2 edition of your software by creating the same source code and labeling it as v2 (or you can modify the "Hello, World!" string to "Hi, GKE!" before constructing a picture): docker build-t gcr.io/${PROJECT ID}/hello-app: v2.

Push the file to the Google Container Registry: docker pushgcr.io/${PROJECT ID}/hello-app: v2 Then add a rolling upgrade to the latest deployment with an object patch: kubectl set the template deployment / hello-web hello-app= gcr.io/${PROJECT ID}/hello-app: v2 View the database again at http:/[EXTERNAL IP] and note the improvements you have made.

Cleaning up To avoid incurring charges on your Google Cloud Platform account for the services used in this guide: Upon finishing this tutorial, take certain steps to remove the following resources to prevent unnecessary charges from incurring on your account: Disable the Service: This phase should fix the Cloud Load Balancer generated for your Service: Kubectl disable the Hello-Web service

This session shows you how to pack a web application in a Docker container image and run that container image on a Google Kubernet Engine cluster as a load-balanced set of replicas that can be scaled to the needs of your users.

Objectives To package and deploy your application to GKE ,

You must: package your app to a Docker image Run a container locally on your machine (optional) Upload the image to a registry Create a container cluster Deploy your app to a cluster Expose your app to the Internet Scale Deploy a new version of your app Before you start Take the following steps to enable Kubern.

Create or choose a project.

Wait until the API and related services are available. It can take a few minutes.

Make sure the billing is activated for your Google Cloud Platform venture. Learn how to ensure your billing is allowed for your venture.

Choice A: Using Google Cloud Shell You should adopt this tutorial using Google Cloud Shell, which comes pre-installed with the gcloud, docker, and kubectl command-line software used in this tutorial. If you are using Cloud Shell, you don't need to download these command-line utilities on your workstation.

Go to the Google Cloud Platform Marketplace to use Google Cloud Shell.

Select the Enable Cloud Shell key on the edge of the console screen.

_____

Google Cloud Platform Console

A Cloud Shell session starts in a new frame at the bottom of the console and shows a command-line request.

Virtual Shell Session Option B:

Using command-line software locally Unless you want to adopt this instruction on your workstation, you will need to download the following tools: Download the Google Cloud SDK, which contains the gcloud command-line application.

Download the Kubernetes command-line software using the Gcloud command-line interface. Kubectl is used to connect with Kubernetes, the container orchestration framework for GKE clusters: gcloud modules download Kubectl Download Docker Community Edition (CE) on your workstation. This will be used to create a container picture for the software.

Install the Git Source Control Tool to get a sample application from GitHub.

Step 1 : Build a GKE image container that accepts Docker images as an application deployment format. You need an application and a Docker file to build a Docker image.

For this tutorial, you will deploy a sample web application called Hello-App, a web server written in Go that responds to all requests with the "Hello, World" message."It's on port 80.

The application is packaged as a Docker image, using the Dockerfile that contains instructions on how to build the image. You are going to use this Dockerfile to prepare the software.

Run the following commands to download the hello-app source code: git clone https:/github.com/GoogleCloudPlatform/kubernetes-engine-samples cd kubernetes-engine-samples / hello-app

Set the PROJECT ID environment variable to your GCP project ID. This variable will be used to link the container image to the Container Registry of your project.

Import PROJECT ID=[PROJECT ID] To create a container picture of this request and mark it for upload, run the following command: docker construct-t gcr.io/${PROJECT ID}/hello-app: v1.

This command instructs Docker to create an image using the Dockerfile in the current folder and to mark it with a name, such as gcr.io/my-project/hello-app:v1. The gcr.io prefix applies to the Google Container Registry where the picture is stored. Running this command doesn't upload an image yet.

You may run docker images command to verify that the construct was successful: docker images Output: REPOSITORY Mark IMAGE ID CREATED SIZE gcr.io/my-project/hello-app v1 25cfadb1bf28 10 seconds ago 54 MB

Step 2: Upload the container image You need to upload the container image to a registry so that GKE can download and run it.

First, configure the Docker command-line tool to authenticate to the Container Registry (you only need to run it once): gcloud auth configure-docker You can now use the Docker command-line tool to upload the image to your Container Registry: docker pushgcr.io/${PROJECT ID}/hello-app: v1

Step 3: Run your container locally (optional) to test your container image using your local Docker engine. Otherwise, open a new terminal window (or a Cloud Shell tab) and run to check whether the container operates and reacts to requests with "Hello, World!": curl http:/localhost:8080 When you've seen a positive response, you can shut down the container by pressing Ctrl+C in the page where docker run command is running.

Step 4: Build a database cluster Now that the container image is saved in a server, you need to build a container cluster to operate the container picture. A cluster consists of a group of Compute Engine VM instances operating Kubernetes, the open source cluster orchestration framework that drives GKE.

Once you have built a GKE cluster, you use Kubernetes to deploy applications to the cluster and control the applications ' lifecycle.

Set the project ID and Compute Engine area options for the gcloud tool: gcloud config set project $PROJECT ID gcloud config set compute/zone [COMPUTE ENGINE ZONE]

Run the following command to build a two-node cluster called hello-cluster: gcloud container clusters create hello-cluster —num-nodes=2 This may take many minutes for the cluster to be created. Once the order has arrived

Output: NAME ZONE MACHINE TYPE PREEMPTIBLE INTERNAL IP EXTERNAL IP STATUS gke-hello-cluster-default-pool-07a63240-822n us-central1-b n1-standard-1 10.128.0.7 35.192.16.148 RUNNING gke-hello-cluster-default-pool-07a63240-kbtq us-central1-b n1-standard-1

10.128.0.4 35.193.136.140 RUNNING

Step 5: Deploy your application To deploy and manage applications on a GKE cluster, you need to communicate with the Kubernetes cluster management system. Typically, you do this by using the command-line kubectl tool.

Kubernetes describes systems as Pods, which are units containing a container (or a set of tightly coupled containers). The Pod is the smallest unit to be deployed in Kubernetes. In this tutorial, each pod contains only the contents of your Hello-App.

The Kubectl Deployment command below allows Kubernetes to build a Hello-Web Deployment on your cluster. Deployment handles multiple copies of your software, or replicas, and schedules it to operate on the different nodes in your cluster. In this case, only one Pod of your application will be running the Deployment.

Run the following command to deploy your application: kubectl create deployment hello-web — image= gcr.io/${PROJECT ID}/hello-app: v1 To see a Pod created by Deployment, run the following command: kubectl get pods Output: NAME READY STATUS RESTARTS AGE hello-web-4017757401-px7tx 1/1 Running0 3s Step 6: Expose your application to the Intern. You must directly open your software to Internet traffic, execute the following command: kubectl expose deployment hello-web — type= LoadBalancer — port 80—target-port 8080 The kubectl expose command above generates a Server tool that offers networking and IP access to the Pods of your request. GKE must build an additional IP and a Load Balancer (subject to billing) for your request.

CHAPTER SIX

Kubernetes Monitoring

With Applications Manager's Kubernetes Management System, visualize the performance of your entire container infrastructure at a glance and investigate issues quickly.

Kubernetes (or k8s) is an open-source container orchestration system for automating the deployment, scaling and management of application containers across host clusters.

Kubernet clusters span across public, private, or hybrid clouds. K8s orchestration allows users to build application services across multiple containers, plan those containers across clusters, scale those containers, and manage the health of those containers over time. This requires Kubernets to monitor the application with a dedicated Kubernetes monitoring solution to maintain the health and availability of the system.

Perceptive Kubernet monitoring tools, such as Applications Manager Kubernet monitoring software, allow administrators to adapt various Kubernet cluster monitoring strategies to account for new infrastructure layers (when adopting containers and container orchestration) with a Kubernets distributed environment.

Monitor Kubernets cluster and gain insights with Applications Manager Kubernets monitoring tool With Kubernets cluster monitoring capabilities offered by Applications Manager, you can: Auto-discover parts and map relationships between cluster objects-Kubernets nodes, namespaces, deployments, replica sets, pods, and containers.

Monitor Kubernet cluster stats and easily identify faults and their sources.

Gain visibility on the operating data, such as the number of resources used, the namespaces per cluster and per pod.

Track the capacity and resource utilization of your cluster and drill into specific cluster parts.

Monitor the installations and upgrades of the software.

Kubernetes Track Resource Consumption Monitor Kubernetes nodes and resource consumption Identify if you have enough nodes in your cluster and the allocation of resources to installed nodes is appropriate for apps to be deployed.

Ensure that all cluster nodes are healthy, monitor the CPU and the Kubernet nodes (workers and masters) memory.

Make informed decisions when defining the number of instances in a node for each pod, including backup instances.

Know whether your clusters are over-or under-provisioned at any given time.

_____

Kubernetes Node Space Use

Ensure pods are accessible and operating Using Applications Manager Kubernetes Management Software, you'll be able to: Verify that all the pods you want are working in the deployment and not in the reboot process.

Identify asset constraints or design faults due to which pods may be absent.

Monitor peaks in resource consumption and know how often requests for all containers on a single node struggle.

Set up container restart warnings to identify issues with containers or their hosts that would affect the performance of their applications.

_____

Kubernetes Pod Memory Usage & Utilization Kubernetes-hosted applications and services The Kubernet monitoring solutions offered by Applications Manager allow you to: monitor the performance outputs of Kubernetes-hosted applications running within your cluster and track any individual errors.

Track the number of network requests sent across containers to different nodes within a distributed service, often from locations around the world.

Manage services to ensure that your deployed applications are always running optimally.

View the status of the Kubernet master and node components API Server, the Etcd key / value store, the Scheduler and the Controller.

Monitor the persistent storage volume that pods consume and the persistent volume claims that it grants exclusive use to storage pods.

Kubernetes Services Set up alerts and generate reports Applications Manager Kubernetes performance monitoring software brings system level metric alerting capabilities so that you can quickly troubleshoot issues on the most important parts of your cluster. You can generate data-rich reports on all important performance attributes to analyze historical trends and make informed choices.

Top 10 Open Source Monitoring Tools for Kubernetes

With over 58K stars on GitHub and over 2,200 contributors worldwide, Kubernetes is the de facto standard for container orchestration. Although discussing some of the main problems involved in running decentralized microservices, a number of new ones have also been added. Not surprisingly, when asked, the engineers list monitoring as one of the main obstacles to the adoption of Kubernetes. After all, the monitoring of distributed environments has never been easy, and Kubernetes adds additional complexity. What is not surprising, too, is the development of various open-source monitoring tools for Kubernetes to help overcome the challenge.

These tools address the different aspects of the challenge. Many support with reports, many aid with measurements. Some are data collectors, while others provide an interface to operate Kubernets from a bird's eye view. Some are native Kubernets, others are more agnostic in nature. Such range and scope testify to the power of Kubernetes as an architecture and a culture, and in this article we will take a look at some of the most common open-source tools available.

Monitoring Kubernets with New Relic Until Kubernetes took over the world, cluster managers, DevOps programmers, application developers and running teams had to conduct a variety of manual tasks in order to plan, execute and maintain their containerized systems. The rise of the Kubernetes Container Orchestration Platform has changed many of these responsibilities, but these same teams now have new things to worry about.

In turn, when Kubernetes solves old problems, new ones can also be developed. Specifically, the adoption of containers and container orchestration requires teams to rethink and adapt their monitoring strategies to reflect the new infrastructure layers introduced in the Kubernets Distributed Environment. You need to monitor: Clusters: Track the capacity and resource utilization of your cluster and be able to drill into specific parts of the cluster, including: Nodes: Monitor CPU, memory, and disk utilization for Kubernet workers and masters to ensure that all nodes are healthy.

Deployments / pods: Ensure that all desired pods are running and healthy during deployment.

Containers: CPU monitor and memory consumption, and how close it is to the limits you've configured. Search for containers that can not be started because they are trapped in a crash loop backoff. Applications: Track the quality and accessibility of applications running within the Kubernets cluster. Measure your request rate, throughput, error rate, and more.

End-user experience: track and monitor mobile application and browser performance to gain insight into such things as response time and error. Monitor loading time and availability to ensure customer satisfaction.

Supporting Infrastructure: Track Kubernet clusters operating through different platforms, including simple on-premise Kubernetes, Amazon Web Services (AWS), Microsoft Azure Cloud, Google Cloud Platform (GCP) and IBM Cloud, as well as managed services such as Amazon Elastic Container Service for Kubernetes (Amazon EKS), Google Kubernetes Engine (GKE), Microsoft Azure Kubernetes Server (AKS), and

This guide was designed to highlight the basics of what you need to learn to successfully track Kubernet's deployments. Speak of it as the Kubernet 101 Tracking.

Monitoring Kubernet clusters In addition to providing access to running information such as the number of resources used and namespaces per cluster and per pod, the implementation of New Relic Infrastructure Kubernets often helps you to see the interaction between artifacts in the cluster by taking advantage of Kubernet's built-in labeling system.

Get started monitoring the Kubernetes cluster with New Relic The New Relic Platform Kubernetes implementation adds in system-level analytics, enabling you to track, troubleshoot, and report on the most important parts of your cluster. Use the out - of-box configuration dashboard to examine a single container or scale up to see massive, multi-cluster implementations through various Kubernet organizations, including nodes, clusters, namespaces, and containers.

To show the Kubernetes integration dashboard: go to infrastructure.newrelic.com> Integrations> Host Integrations, then pick the Kubernetes dashboard connection.

Mount the Kube-state-meters.

To complete the setup and configuration, follow the New Relic implementation guide.

It unlocks the standard Kubernetes dashboard: Show the information in pre-built dashboards for instant access into your Kubernetes system Build your own unique Kubernetes dashboards using NRQL queries in insights.newrelic.com.

Get a full cluster overview Track and monitor your cluster's capacity and resource utilization.

Because your Kubernet implementation is likely to be operated by different teams, including site reliability engineers (SREs), operational technicians, and programmers, you increasing find it difficult to keep track of the current state of the cluster. Key questions include: How large is my Kubernet deployment?

How many servers, namespaces, clusters, modules, and containers do I manage in a cluster?

How New Relic supports Part of the standard New Relic Infrastructure Kubernet application screen, the cluster summary monitor shows the number of entity classes that apply to you. Think of it as a glimpse, or a curated set, of the items that Kubernet handles in your cluster.

The standard New Relic Infrastructure dashboard provides an overview of the artifacts in each group.

Suggested warnings There is no urgent need to add notifications to the cluster summary. But we recommend that you keep this dashboard visible in your workspace so that everyone on your team can see the current status of your Kubernet deployment.

_____

Track cluster resource utilization If you handle clusters, you need enough available assets in your cluster to avoid running into problems if planning pods or distributing containers. If you do not have enough capacity to meet the minimum resource requirements of all your containers, increase the ability of your nodes or attach additional nodes to spread the workload.

You should know: what percentage of cluster resources you use at any given time If your clusters are over-or under-provisioned How much demand you have placed on your systems How New Relic helps Our Kubernet integration monitors and tracks aggregated core and memory usage across all nodes in your cluster. It helps you to fulfill the resource requirements for optimum application performance.

The standard New Relic Technology application of key and storage use.

Suggested Warning Set warnings to the cores and space use of the hosts in your group.

Track node resource utilization Besides simply keeping track of nodes in your cluster, you need to track CPU, storage, and disk use of Kubernet nodes (workers and masters) to insure that all nodes in your cluster are safe.

Use this information to ensure: You have enough nodes in your cluster The resource allocation of current nodes is adequate for deployed apps You are not breaching any resource caps, etc. It is stable How New Relic lets New Relic monitor resource consumption (used cores and memory) for each node in Kubernets. It helps you to track the number of network requests that have been sent through containers to various nodes within a distributed system. You can also track resource metrics for all containers on a specific node — regardless of which service they belong to: the default New Relic Infrastructure dashboard to monitor Node Resource Consumption.

Please ensure that your existing implementation has sufficient resources to scale. You don't want new node deployments that are blocked by a lack of resources.

Suggested Warning Set notifications such that you will be alerted if the host ceases communicating or if the CPU or storage consumption of the node falls below the target level.

Check for missed pods From time to time, you can notice that your cluster is lacking a pod. A pod may be absent if the engineers did not provide

sufficient resources when it was planned. The pod may never have started; it's in the reboot loop; or it's gone missing due to an error in its setup.

To ensure that Kubernetes does its job properly, you need to confirm the health and availability of pod deployments. The Pod Deployment determines the number of instances to be available for each pod, including the backup instances. (This is referred to as the ReplicaSet in Kubernetes). Sometimes the number of active pods on each deployment is not defined in the Replicas sector. Even if this is the case, Kubernetes can decide if it can run another instance based on the assets the administrator has specified.

Kubernetes pod deployments How New Relic helps New Relic make it easier to avoid this problem by knowing the limitations of the cluster resource.

If you do not have enough resources to assign a batch, add more container instances to the cluster or swap a container instance for one with the correct amount of resources. In general, you can use the New Relic Kubernets interface to track missed pods and immediately identify applications that require attention. This often creates an opportunity to address asset or functionality problems until they impair the quality or efficiency of the software.

Standard New Relic Technology dashboard to track missed pods through launch.

Suggested warning Set an alarm when the missing pod price of the deployment falls above a certain level for a certain period of time. If the number of available deployment pods drops below the number of pods you listed when you built the deployment, the alarm will be activated. The alert will be applied to any deployment that matches the filters you set.

Find pods that do not operate Kubernets dynamically assign pods to a cluster; if you have resource issues or configuration failures, scheduling would probably fail. If the pod isn't working or even planned, there's a conflict with either the pod or the group, or with all of the Kubernet deployment.

When you see that pods are not working, you'll want to know: If there are pods in the reboot process How often requests fail If there are resource problems or configuration errors Why New Relic supports As noted, if you have resource issues or configuration errors, Kubernetes may not be able to schedule pods. In such instances, you would like to test the safety of your applications and recognise design bugs or resource problems.

With the New Relic Framework Kubernet implementation, you can use default deployment information to discover and monitor pods that may not be working and group them by cluster and namespace.

With the New Relic Framework Kubernet implementation, you can use default deployment information to discover and monitor pods that may not be working and group them by cluster and namespace.

Suggested Warning Set updates to the state of your pods; warnings should be activated when the pod has the status of "Failed," "Pending" or "Missing" for the period of time you decide.

Troubleshoot Container restarts Containers should not be restarted under normal conditions. Container restarts are a sign that you are likely to hit a memory limit in your containers. Restarts may also indicate a problem with either the container itself or its host. In addition, because of the way Kubernets schedules containers, it can be difficult to resolve container resource issues because Kubernets will restart — if not kill — containers when they reach their limits.

Monitoring container reboots helps you understand: If any containers are in a restart loop How many container reboots happened in X cycles Which containers are rebooting What New Relic supports A continuous container reboot count becomes part of the standard container information New Relic collects with the Kubernet implementation.

Reboots the default New Relic Infrastructure dashboard to monitor the container.

Suggested notification This is an alternative alarm situation. Kubernetes immediately restarts containers, so setting up an alert can send you instant, helpful warning, but don't let the container restarts deter you from sleeping.

Watch Container Resource Use Tracking Container Resource Use lets you insure that their containers and apps remain healthy. For eg, if a container reaches its storage cap, the kubelet agent could destroy it.

While tracking the use of the container resource, you'll want to know: if your containers reach storage limitations and impact the output of their applications If there are resource consumption peaks If there is a trend for the distribution of errors per container How New Relic supports

Next, define the minimum amount of CPU and storage that the container needs to run — which must be provided by the cluster— and track certain assets with New Relic.

Second, track the limits of the container capacity. These are the total amount of resources that the container can consume. In Kubernetes, the asset caps are unbound by definition.

The standard New Relic Infrastructure dashboard to track the storage use of the tank.

Check The blue line reflects the maximum amount of the asset that the container is permitted to ingest. The yellow line indicates the minimum amount of CPU or storage that the container needs to run, which must be assured by the device. This form of control will help to resolve resource consumption problems proactively until they impact the request.

Suggested alerting Set alerts on container CPU and memory usage and on limits for those metrics.

Monitoring applications running in Kubernets The key advantage of Kubernetes is that it decouples the software and its business logic from the specific details of its runtime environment. This ensures if you ever have to change the underlying infrastructure to a different Linux edition, for instance, you won't have to completely rewrite the program software.

While tracking programs operated by an orchestration framework, being able to relate a software failure track, for example, to the box, pod, or host it's running in can be very helpful for debugging or troubleshooting.

You need to track the quality and availability of applications running inside the Kubernetes cluster on the application layer. You do this by measuring statistics such as request speed, latency, and error rate.

Monitoring applications running in Kubernets Get started monitoring applications in Kubernets New Relic APM lets you add custom attributes and that metadata is available in transaction tracks collected from your application. You may create custom attributes to collect information about the specific Kubernetes node, pod, or namespace where a transaction happened.

You will need to integrate your software with the Kubernetes Downward API to get started monitoring applications running on Kubernetes. We have built a demo program to demonstrate how this operates in the Node.js framework — fork this project for your own use. (Our Tracking Application Quality in Kubernetes blog post discusses how to apply this kind of Kubernet metadata for APM-monitored application transactions.) Trace Transaction Tracks As you run apps in Kubernetes, system containers also travel across the cluster when instances scale up or down.

This scheduling is done automatically in Kubernetes, but could affect the performance or availability of your application. If you are an application developer, it is important to be able to correlate Kubernetes objects with applications for debugging and troubleshooting.

You'll want to know: Which apps are connected with which group How many transactions are happening within a specified pod As New Relic allows to manage transaction monitoring in Kubernets, you need a code-centric view of your applications.

You need to associate the programs with the box, pod, or host in which it is operating. You also need to identify pod-specific performance issues for the workload of any application.

Transaction attributes display the hostname and IP address of Kubernets where an error occurred Knowing the names of the pod and the node where an error occurred may speed up the troubleshooting process. Visibility of transaction traces quickly highlights any abnormalities in your Kubernetes-hosted application.

To learn how to make payment tracks more transparent in Kubernetes, see the blog post, Tracking Transaction Quality in Kubernetes.

Suggested Warning You will want to set up reminders for all applications running in development. Specifically, you will want to respond to API service requests, payments, server latencies, uptime, and throughput, and send alerts when any of these parameters fall below the thresholds you identify.

Prevent bugs If a single pod or a specific IP pod starts to fail or throw bugs, you need to fix bugs before those bugs damage the cluster or program. When something goes wrong, null triggers origin as quickly as possible.

You'll want to know: in which namespace / host / pod the transaction failed If your system is running as planned in all pods The output of the X program running on pod Y How New Relic works with New Relic, you can get a code-centric view of the applications running within your cluster, check the Kubernets-hosted quality outlier apps, and track any individual errors.

APM Error Profiles can automatically detect when errors occur within the same pods and from the pod IP addresses Tracking System Quality in Kubernet's blog post will help you find applications that might trigger performance problems in your cluster.

Suggested Warning Set warnings to monitor error rates for any applications running in Kubernet production environments.

Monitoring end-user experience while Kubernetes is operating When you order a product from a delivery service and it arrives defective or late at your house, do you really know which aspect of the delivery process has failed? Whether it was the responsibility of the supplier, the retailer or the delivery service, the end result is just as irritating.

The same logic applies to businesses running applications in Kubernetes: if the user navigates to their platform and the website does not launch or takes too long, the consumer is not involved in the technical reasons for this. That's why it's not enough to watch the output of your own

processes, it's also important to control the front-end quality of your apps and learn what your consumers are actually experiencing.

Monitoring end-user experience when operating Kubernetes Even though your application is running in Kubernetes, you can still define and monitor key indicators of customer experience and figure out how the phone or web quality of your application affects your company.

Why New Relic works For instance, once developers first move to Kubernets, they should set up a pre-migration benchmark to measure the average load time of their front-end software before and after migration. Developers can use the same techniques outlined in the Kubernet blog post of Tracking Application Performance to gain insight into key indicators such as response time and error for mobile applications and device quality. It is also essential to monitor load time and availability in order to ensure customer satisfaction. New Relic Browser and New Relic Mobile are designed to give you a crucial insight into the experiences of your users.

The New Relic App History page shows a rundown of app quality for that device Quickly see crash events, app updates, and more with New Relic Mobile In fact, users and operators both need to consider the accessibility of any Kubernet-hosted service, often from locations around the world. New Relic Synthetics is designed to track the availability and performance of applications from a wide variety of locations.

New Relic puts together business-level information and quality details in a single glass window. It allows consumers from growth, production, consumer and customer support groups to identify potential places of quality enhancement and to find better ways to correct mistakes that may impact individual customers.

Suggested Alerts New Relic App Alerts: Mobile Network Error Level and Response Time to insure that you are alerted at the most important endpoints New Relic Web Alerts Browser Session Count fall to suggest availability issues User Javascript Error Rate or Error Count Browser Interaction Length New Relic Synthetics Alerts: Synthetic Error Rate or Error Count?

Synthetic response times?

Synthetics ping tests Metrics such as page load time, phone crashes, JavaScript bugs, ping reviews, and script checks through a main client system.

Monitoring Kubernet support infrastructure Major cloud providers (AWS, Azure, GCP, and others) are now offering Kubernetes-as - a-Service platforms. Unfortunately, due to Kubernet's dynamic scheduling, without proper monitoring in place, it may be difficult to diagnose failure points or track other issues on these cloud platforms.

Monitoring Kubernet service infrastructure While operating Kubernets in a cloud environment, you'll want to know: If your cluster is spread around multiple regions If a container is running on a vendor's network or on-site at the time of shutdown, How New Relic helps New Relic lets you monitor, search, and warn utilization statistics and failures inside your cloud environment; Search for statistics to collect from all levels of your network-based Kubernet framework, even if the clusters are distributed through multiple data centers or cloud providers.

Search for statistics to collect from all levels of your network-based Kubernet framework, even if the clusters are scattered around multiple data centers or cloud providers.

Suggested alerting Set alerts to track service latency, uptimes, and throughput for applications running on any Kubernetes-as-a-Service platform.

Amazon Elastic Container System for Kubernetes (Amazon EKS) Amazon EKS offers Kubernetes as a managed service on AWS. This will make it easier to install, handle and scale containerized software on Kubernets.

How New Relic supports Though EKS makes it easier to install and operate Kubernetes, but because of the ephemeral nature of Kubernetes-based workflows, customers require specialized cluster, node, pod, container and system level control. New Relic gives that crucial visibility to your clusters.

Your Amazon EKS system contains thousands of marks and identifiers linked to your devices, bins and microservices. New Relic automatically collects these labels and tags so that you can view all of your Kubernets entities more easily, no matter how you arrange them.

Read more: Amazon EKS Is Here— and You Can Track It With New Relic Google Kubernetes Engine (GKE) GKE offers a framework for installing, maintaining, and scaling the containerized software leveraging Google's infrastructure.

How New Relic supports as GKE handles the Kubernet network, New Relic allows troubleshooting, balancing and handling the dynamic environment simpler. New Relic helps ensure that your clusters are operating as planned and can help you easily spot performance issues inside the cluster until they impact your customers.

Learn More: Track Google Kubernetes New Relic Microsoft Azure Kubernetes Server (AKS) AKS manages the managed Kubernetes system, making it easier to install and maintain containerized systems without container orchestration experience. It also removes the strain of continuing operations and maintenance by delivering, updating, and expanding assets on demand without having the systems offline.\How New Relic works If you are operating containers in AKS, use the New Relic software and infrastructure-centric views to easily see all of the implementations through

AKS.

Write the deployment implementation guide of Kubernetes.

RedHat OpenShift OpenShift provides developers with an Integrated Development Environment (IDE) to create and deploy Docker-formatted containers and then control them with the Kubernetes Cluster Orchestration Framework.

Why New Relic makes you always want to learn that programs are running on which nodes in your Openshift system. New Relic lets OpenShift administrators see which applications are stopped and why. New Relic also makes it easy to assign programs and their services to namespaces and then track namespaces.

Read the integration documentation for OpenShift.

Toolset The new DevOps toolkit New Relic is just one of the many essential tools you'll need for the DevOps project.

CHAPTER SEVEN

The Best Open-Source Tools For Kubernetes Monitoring

Distributed computing and orchestration have solved many problems, but they have also created new challenges. While a Kubernet cluster tends to a client to be a single device, it is actually a set of separate nodes and multiple resources that have been linked.

With this new way of designing and managing apps, the management and enforcement methods need to be changed — and so will the resources you use. Here are the most common and powerful open-source monitoring tools you can choose from while operating with Kubernetes.

Learn how to change the IT with AIOps in the TechBeacon Guide. Plus: Download an analyst's report on how AI is transforming the position of IT.

1. Kubelet In the Kubernet cluster, Kubelet serves as a link between the master and the nodes. It's the primary node agent which operates on each node and manages a collection of pods. Kubelet tracks PodSpecs via the Kubernetes API database and gathers resource consumption stats and pod and activity status.

Kubelet collects statistics on individual container usage from Docker's Container Advisor (cAdvisor). Yet Kubelet often embraces PodSpecs offered through a number of processes and assures that the containers listed in those PodSpecs are up and running. These aggregated pod resource usage statistics are shown in the REST API.

1. Container Advisor (cAdvisor) cAdvisor is a container asset use and performance analysis tool that is built into the Kubelet binary. CAdvistor auto-discovers all containers on a computer and gathers data on storage, network use, file system, and CPU usage. CAdvisor has native support for the Docker server. It does not operate at the pod stage, but at each node.

Nonetheless, be advised: cAdvisor is a simple-to-use but restricted device, so if you are trying to store statistics for long-term use or conduct complicated tracking behavior, cAdvisor will not meet your needs.

CAdvisor DashboardcAdvisor Dashboard shows live data about all containers on a specific system.

Enterprise Service Management introduces creativity to the business. Read more about TechBeacon's latest ESM guide. Plus: Get the 2019 ESM Forrester Splash.

1. Kube-state-metrics Kube-state-metrics listens to the Kubernetes API server and generates state-of - the-art metrics for multiple Kubernet objects, including cron jobs, configure maps, pods, and nodes. Such metrics are unmodified, unlike kubectl metrics that use the same Kubernet API but use some heuristics to view comprehensible and understandable messages.

Kube-state-metrics uses the Golang Prometheus application to export metrics to the Prometheus Metrics Exhibition Framework and show metrics to the HTTP endpoint. Prometheus can consume the endpoint of the web.

This method is not oriented towards efficiency and wellbeing, but rather towards cluster-wide, state-based measurements such as the amount of required pod replicas for deployment or the maximum CPU resources available on the node.

1. Kubernetes Dashboard Kubernetes Dashboard is a web-based, UI add-on to Kubernetes clusters. It has many features that allow users to create and handle workloads as well as exploration, load balancing, setup, processing, and monitoring. It's helpful for small clusters and for people starting to learn Kubernet.

This method offers different viewpoints of the CPU and storage consumption statistics aggregated across all nodes. It may also be used to monitor the health status of workloads (pods, deployments, replica sets, cron jobs, etc.). Installing the Kubernet Dashboard is quite simple and can be achieved with ready-to-use YAML software.

Kubernetes Dashboard is a general purpose online user interface for Kubernetes clusters that provides governance, troubleshooting and tracking functionality.

1. Prometheus Prometheus is one of the most common monitoring tools used by Kubernetes. It is community-driven and a member of the Cloud Native Computing Foundation. This project, first developed by SoundCloud and then donated to the CNCF, is inspired by Google Borg Monitor.

Prometheus preserves all of its information as a time series. This data can be queried through the PromQL query language and presented with the built-in speech browser. Since Prometheus is not a dashboard, it relies on Grafana for data visualization.

Version 1.0 of this tool was released in 2016 and is becoming one of the most widely used Kubernet monitoring tools. Other Kubernet ecosystem tools, including Istio, include the built-in Prometheus adapter, which displays the generated metrics.

You can download Prometheus directly as a single binary that you can operate on your host or as a Docker server. Running Prometheus at the top of the Kubernetes can be easily accomplished with the Prometheus Operator.

1. Jaeger Jaeger is a tracking system launched by Uber Technologies; it is used for troubleshooting and managing transactions in dynamic distributed systems.

As microservices and distributed systems rise, problems may include the spread of context, the monitoring of distributed transactions, and the optimization of latency. Jaeger discusses these issues as well as others that can be encountered in distributed systems.

Jaeger has native support for OpenTracing and tackles two main areas: networking and observation.

Implemented using React, the Jaeger UI provides a minimalist user interface designed for displaying large data volumes.

1. Kubewatch Kubewatch is a Kubernet watcher that publishes activity updates on a Slack list. This tool allows you to specify the resources that you want to monitor. It's written in Golang and uses the Kubernetes client library to communicate with the Kubernetes API database.

Using a basic YAML folder, you can choose the tools you want to monitor, including database sets, servers, pods, clone sets, replication controls, utilities, passwords, and configuration charts.

1. Weave Scope Weave Scope is a zero setup management software developed by Weaveworks. This creates a chart of processes, servers, and hosts in the Kubernetes cluster to help you understand Docker containers in real time. It can also be used to handle containers and to run test commands on containers without leaving the visual UI.

If you are searching for a functional visualization method to get a visual summary of your Kubernet cluster— including the software, network, and communications between your cluster nodes— Weave Scope will support. It device is extensible with a few plugins.

Weave Scope lets developers view real-time dashboards and background statistics, labels, and container-related metadata.

1. EFK Stack The EFK stack comprises of Fluentd, Elasticsearch and Kibana.

Both methods function well together and together provide a reliable solution for tracking and log processing of Kubernets.

Fluentd gathers logs from pods operating on cluster nodes, then forwards them to a consolidated Elasticsearch. Elasticsearch then ingests these logs from Fluentd and stores them in a central location. It is also used to search text files efficiently.

Kibana is the UI; the client can imagine logs and statistics obtained and build custom query-based dashboards.

The EFK platform is helpful for troubleshooting files, dashboarding, and tracking issues as they arise— all in a user-friendly interface.

Choose the right tool for the successful monitoring of kubernetes!

Kubernetes is a production-ready, open-source framework developed with Google's expertise in container orchestration, connected to the best-of-breed ideas of the community. It is expected to simplify the installation, scaling and service of software containers.

With the modern way of building and running applications, your control and monitoring strategies need to be advanced and the tools that you use. Standard network monitoring tools may not be adequate and you need a customized Kubernet monitoring system, as set out below.

Many support with reports and some aid with measurements. Many provide an app to run Kubernetes from a bird-eye view. Some are true Kubernets, while others are more agnostic.

Prometheus: Prometheus is one of the most common and strongest monitoring tools used by Kubernetes. It platform was developed early by SoundCloud and then contributed to the CNCF. Google Borg Monitor encourages you.

Prometheus Ok, Prometheus preserves all of its information as a time series. In brief, Prometheus stands out among other time-series servers, its built-in alerting systems, its multidimensional information system, its pull vs. push framework, its PromQL (Prometheus querying language) and, of course, its ever-growing community.

Some more characteristics of Prometheus include: No dependence on distributed storage; Targets are discovered via network discovery or static configuration of PromQL, a dynamic query language to gain from this dimensionality Single database nodes are stand-alone Time series compilation happens via a pull template over HTTP Pushing Time series is enabled through an intermediary A multidimensional gateway. They have great documentation, but if you're looking for video-based learning, check out this Udemy course.

1. Kubewatch

Kubewatch is a Kubernet watcher that publishes activity updates on a Slack stream. This tool gives you the facility to determine the resources you need to monitor. It is created in Golang and uses the Kubernetes client library to connect to the Kubernetes API server. This library acts as the foundation for the analysis of the Kubernets case.

The Kubewatch kubewatch is easy to customize and can be installed either with a helm or with a device installation. More specifically, kubewatch will see the updates that you need to monitor for unique Kubernet resources — deployments, node sets, pods, servers, replica sets, utilities, replication controllers, keys, and configuration charts.

1. Jaeger Distributed tracking continues to grow in the control and troubleshooting of Kubernetes ecosystems. Jaeger is a monitoring device launched by Uber Technologies. It is used for monitoring transactions and troubleshooting in complex distributed systems.

Jaeger supports Java, Python, Node, and C++ OpenTracing-based instrumentation. It uses robust upstream testing of individual service / endpoint probabilities and embraces multiple storage backends— Cassandra, Elasticsearch, Kafka, and memory.

Some of the other capabilities of Jaeger include: Distributed Transaction Tracking Distributed Context Propagation Performance / Latency Optimization Root cause assessment System Dependency Assessment 4cAdvisor cAdvisor is configured for storing, sorting, and distributing service use and output data regarding operating containers. It has also been translated into Kubernets and incorporated into the Kubelet binary. It's simple to use (exposes Prometheus out - of-box metrics) but not robust enough to be recognized as an all-round monitoring solution.

CAdvisor In addition to the others, cAdvisor is not implemented per pod but at the node stage. Auto-determines all containers operating on the device and gathers machine statistics such as ram, CPU, network, etc. cAdvisor is a basic tool, and the following are some of its functions.

Native support for Docker containers and other types of container aid.

Supports the export of stats to various storage plugins, e.g. lnfluxDB, etc., makes the cumulative use of the machine by evaluating the root container on the computer.

Support for running standalone outside the Docker or any other container as well.

CAdvisor is running per node. Auto-discovers all containers in a specified node and gathers Ram, file system, and network consumption statistics.

Metrics can be viewed on the Web-UI, which exports live information on all containers on the system.

5Cabin Cabin is the perfect native mobile dashboard device for the Kubernets. Cabin UI was built using React Native and therefore operates on iOS and Android apps. It's on the move assistant, which gives fine-grained actions to manipulate Kubernet's resources. The Cabin app is a touch-advance app.

You can also delete pods with a simple left swipe, for example. You can also level the rollouts with a finger tap.

Cabin Some other features: Build Specific Deployments Scale Deployments and Replication Controllers Swap Server Types Expose Deployments via GKE integration resources for single-click cluster distribution Access logs in several containers Delete and attach Open NodePort products labels to your app Execute commands in 6Telepresence Containers Telepresence enables you to operate a specific servicel It allows developers operating on multi-service operations to follow every locally available software to check / debug / edit the site. You can run a debugger or an IDE, for example.

Telepresence

It also allows developers to quickly develop a particular service globally, even if it relies on separate services in the cluster. Make a transition to your service, save it, and you can instantly see the new service in action.

Telepresence is an impressive local development environment for services operating in Kubernetes. The live debugging component is new and has grown very rapidly. Below are some of its other features.

Enable the software in the container to connect to an IDE or a debugger running on the server.

Telepresence utilizes an OpenShift-specific proxy picture while viewing an OpenShift group.

Telepresence frequently facilitates the movement of traffic to and from other containers in the tank.

Telepresence uses the Docker-accessible file as a backup directory.

Weave Scope

Weave Scope is a troubleshooting and monitoring tool for Kubernetes. This makes logical topologies of your software and network which make it easier for you to understand, monitor and control your containerized, microservice-based system.

Weave Scope offers you a top-down perspective of both the software and the full network. It allows you to identify any problems with your distributed containerized app in real time as it is being deployed to a cloud provider.

Some of the capabilities of the Weave Browser include: support for any deployment type (Local, Hosted, or Hybrid) and ability to capture and monitor Host / Container metrics Apply statistics, incidents, and tags from Kubernets Real-time Context Metrics Nodes can be sorted by CPU and Memory Management so that you can quickly identify containers using the most assets.

Grafana:

Grafana is used to calculate statistics, but it is also used as an alert device. Grafana may send an alert to Slack, Webhook, fax, or alternate channels of communication. The key reason is the origin of your data: Grafana will request several parties at the same time.

You can search from a server like ElasticSearch or track software like Cloudwatch, and you can also set up notifications. Some of the other apps are as follows.

The warning director does the alerting component Quick deployment of exporters The app uses Kubernetes labels to sort pod metrics as well.

The Pod / Container interface uses pod labels to quickly find the relevant container or containers.

_____

9Zabix

With Zabbix, it is possible to build virtually unlimited data types from the system. High-performance real-time monitoring systems which enable tens of thousands of databases, virtual machines, and network devices to be managed simultaneously.

In addition to saving data, visualization features are accessible as well as extremely flexible ways to identify data for alarming purposes.

Some of Zabbix's capabilities include: Root Cause Analysis Zabbix helps keep information in JSON format, so many developers may use it as well.

Real-Time Tracking Zabbix proxy is highly recommended for large-scale production systems.

Drill-Down Reports Low-level exploration automatically checks the latest nodes without any complications.

Highly configurable and expandable.

CHAPTER EIGHT

How To Secure Kubernetes

When companies drive the introduction of container and database orchestrators, they will need to take the necessary steps to secure such a critical part of their computer infrastructure. To assist in this effort, check out these nine Kubernetes safety best practices, based on customer feedback, you can adopt to help protect the infrastructure.

Update to Current Version Additional safety features— and not just bug fixes — are applied to every quarterly update, and to make the most of them, we suggest that you use the current stable version. The best thing to do is to run the latest update with the latest updates, especially in the light of the discovery of CVE-2018-1002105. Upgrades and maintenance can become more complicated the further you slip, but aim to update at least once a half. Using a controlled Kubernetes provider can render updates quite simple.

Allow Role-Based Access Regulation (RBAC) Control of who can access the Kubernetes API and what privileges they have with Role-Based Access Control (RBAC). RBAC is usually enabled by default in Kubernets 1.6 and beyond (later for some managed providers), but if you've updated since then and haven't changed the setup, you'll want to double-check your settings. According to the manner in which Kubernetes authorization controllers are combined, you will both allow RBAC and disable legacy Attribute-Based Access Control (ABAC).

When RBAC is applied, you still need to make effective use of it. Cluster-wide permissions will usually be resisted in favour of namespace-specific permissions. Avoid giving some cluster admin privileges, even for debugging — it is much easier to grant access only as required on a case-by-case basis. You will discover cluster functions and responsibilities using' kubectl getclusterrolebinding' or' kubectl get role-binding–all-namespaces.'

Quickly verify who is given the special “cluster-admin” role; in this example, it's only the “masters” group: When your request requires access to the Kubernetes API, build system accounts individually and assign them the smallest collection of permissions required for each usage site. This is safer than giving overly broad privileges to the default user for a namespace.

Many apps do not need to use the API at all;' automountServiceAccountToken' can be set to' zero' for these programs.

Using Namespaces to Set Security Limits The development of independent namespaces is an essential first layer of element isolation. They find it much easier to enforce security controls, such as network rules, while different types of workloads are implemented in distinct namespaces.

Is your department utilizing namespaces efficiently? Find out now by searching for any non-default namespaces: Separate Critical Workloads To reduce the potential impact of conflict, it is best to run critical workloads on a separate group of devices. This strategy reduces the risk that a critical program will be reached through a less protected system sharing a runtime container or a server.

For example, a compromised node's kubelet credential will typically reach the contents of the secrets only if they are placed in pods on that node— when sensitive secrets are arranged across the cluster on many nodes, the opponent will have more chances to capture them.

You can accomplish this isolation by using node pools (in the cloud or on-site) and Kubernet namespaces, taints, tolerances, and other restrictions.

Safe Cloud Information Control Sensitive information, such as kubelet admin key, can sometimes be hacked or misused for clustering rights. For example, latest Shopify bug bounty disclosure reveals how consumers have been able to scale rights by trusting the microservice and leaking information from the cloud provider's metadata domain.

GKE's metadata concealment function updates the cluster configuration process to prevent this disclosure, so we advise utilizing it until it is substituted by a permanent solution. Similar counter-measures may be expected in other settings.

_____

Build and Define Cluster Network Policies Network Policy helps you to manage network access to and from your containerized applications. To use them, you'll need to make sure that you have a networking service that supports this resource; with some managed Kubernet services, such as Google Kubernetes Engine (GKE), you'll need to opt in. (Enable network policy in GKE would include a quick roll-up if your cluster already exists.) Once this is in effect, begin with some simple default network policies, such as blocking traffic from other namespaces by definition.

If you are operating in Google Container Engine, you should test if your clusters are running with policy protection enabled: Use a Cluster-wide Pod Security Policy A Pod Security Policy establishes the rules for how workloads are allowed to run in your cluster. Try specifying a rule and implementing the Pod Security Policy Admission Controller— the guidelines vary depending on your cloud provider or installation template. As a start, you might need to drop the NET RAW capability to defeat certain classes of network spoofing attacks.

Harden Node Protection You should follow these three steps to improve the node protection: make sure the host is safe and configured correctly. Another approach to do this is to test the software against CIS Benchmarks; most devices feature an autochecker that will automatically determine conformity with these requirements.

Monitor network connectivity to critical ports. Make sure the network blocks access to kubelet ports, like 10250 and 10255. Consider limiting connections to the Kubernetes API database, except for trustworthy networks. Malicious users have misused connections to these ports to operate cryptocurrency mines in clusters that are not designed to allow authentication and authorization on the kubelet API database.

Minimize operational connections to Kubernet nodes.

Generally speaking, node access in your cluster should be restricted— debugging and other tasks can usually be handled without direct access to the node.

Switch on Audit Logging Make sure that audit logs are allowed and checked for error or unauthorized API calls, particularly any authorization failures— these log entries will have a "Forbidden" status alert.

Authorization failures may indicate that a hacker is attempting to misuse stolen credentials. Managed Kubernet providers, including GKE, provide access to these data in their cloud console, and may allow you to set up notifications for permission failures.

How do I Secure Kubernets Using Cloud SIEM?

Kubernetes, commonly referred to as K8s, is an open-source container management system developed by Google. Containers and technologies such as Kubernetes allow the automation of many facets of software delivery, which offers enormous benefits to enterprises.

K8s are just as vulnerable to attacks and cybercrime as conventional settings, both in public and private clouds. In this blog post, we've gathered everything you need to know to make sure your Kubernet system is secure.

Here's what we're tackling: What are the security challenges of K8s?

Why do you protect K8s anyway?

How to protect a K8s stack How to maintain a full application stack like K8s Security challenges Kubernets Kubernets containers operating in pods can be targeted from outside through the network or internally by insiders. Such insiders may be the targets of phishing attacks, whose networks are platforms for insider attacks.

Such assaults are most often triggered by System mis-configuration. Mis-configured system creates an opportunity for an intruder to get inside the container and its context to start looking for flaws in the network, process controls, or file system.

Container-level problems

Once a container becomes infected, attackers may try to escalate permissions and take control over other containers or the whole cluster. Pay close attention to tank design and overall safety hygiene.

Kubernet infrastructure vulnerability Kubernetes itself is not the only target for hackers. We also try to hack the API database or Kubelets (agents that use PodSpecs to insure that containers are stable and operated according to specifications) which ensures that your Kubernetes safety checklist needs to be extended to include tasks to ensure the security of the entire infrastructure.

Attacking orchestration software specifically helps attackers to interrupt operating programs and even gain control over the services used to manage containers, or even acquire admin access privileges over clusters.

The Database system Kubernetes is completely API-driven, which ensures that it is necessary to control and restrict who can reach the cluster and what activities they are allowed to perform. This is going to be your first line of defense.

Visibility What's more, hackers can actually find the Kubernet clusters on the Web very quickly. For example, Kubernetes uses etcd as its cluster server. It listens to port 2379/TCP, which is indexed by Shodan, and can be easily found.

General exposure

Etcd resources are not encrypted, which makes it very easy for hackers to effectively target your cluster server and even compromise your entire system. The Kubernetes API is usually available while implemented, so security is a top priority. Fortunately, many Kubernet implementations have encryption for this node.

_____

Unauthenticated ports

The Kubelet API used by Kubernetes to handle containers on individual nodes and clusters is unauthenticated. This helps hackers to run software in your containers and possibly take over your whole cluster.

How do I protect the K8s stack?

Surveillance

You can not protect Kubernets without introducing multiple layers of surveillance. Monitor critical vectors to detect any events that may have harmful consequences. In addition, monitor: • the whole network: this is the standard entry point for hackers as it is the first place to break in.

• Containers: this will help you to evaluate a suspect system once it has begun or any effort to scale rights.

• Hosts: it is good to use traditional endpoint security to monitor exploits against kernel or system resources.

Source: kubernetes.io User Access Restrictions Secure API Access through the following: • Transport Layer Security (TLS) for all API traffic: this is what K8s requires by definition • API Authentication: once you download a cluster, choose an authentication scheme that fits typical access patterns. All API clients, including those that are part of the infrastructure, must be authenticated.

• Authorization of the API: every API call should pass an authorization check once it has been authenticated. K8's Role-Based Access Control

(RBAC) controls access to application workloads and K8s system resources. It's important to correctly customize the RBACs.

The Cubelet Access Control Access to this API will not be authenticated by definition.

Rule management Workload / User Resource Management Specific policies control the actions of a system, whether on a network, on its own or other assets. Consider the following policies:

• Resource utilization restriction for clusters: resource allocation limits the number or ability of resources allocated to namespace, although limit ranges restrict the total or minimum size of some of those resources

• Pod security policy: monitor security-related elements of pod configuration by means of a set of conditions that the pod will operate to be allowed into the framework. The app allows you to easily calculate and interpret key data to immediately identify abnormalities. You can perform root cause analysis to improve MTTI and MTTR on a single screen.

The Sumo Logic framework for Kubernetes integrates seamlessly with any container design, whether local, hosted or hybrid. The software automatically runs as a stand-alone device in a data center or as part of a network infrastructure, and incorporates Sumo Logic applications for all major cloud hosting systems, including AWS, Google Network, Microsoft Azure, and VMware Vsphere.

You can find detailed instructions on how to install the app here.

Securing the full application stack, including the K8s RBACs, is not just for API approval. It is necessary for all machine tools to be correctly installed. To do this properly, make sure that you check these access controls and customize them as required:

Restrict Cubelet Permissions Require encryption for all external ports Limit or disable console access Source: kubernetes.io Sumo Logic provides full stack transparency to applications, like Kubernet node orchestration. With the dedicated K8s software, programmers are safe from gathering and tracking data from independent, siloed devices together. Through container software to microservices and apps, Sumo Logic offers detailed graphical perspectives into each level of the system Key takeaways: K8s can be targeted from the outside or from the inside.

Common causes of attacks involve software misconfiguration, container level problems, or Kubernets that can be targeted both from outside and within. Its entire infrastructure is vulnerable to Kubernetes and must be secured because it is fully API-driven, its clusters can be easily traced, and the services used by Kubernetes are not authenticated. Ports are also unauthenticated.

Securing the K8s network, tracking everything and enforcing stringent API access restrictions, restricting access to Kubelet, managing the configuration and client functionality with policies and securing cluster components against compromise To safeguard the full application stack, configure RBACs to restrict privileges, limit console access and authenticate all external ports The highly dynamic design of the container e.

CHAPTER NINE

Configuration Management

Better Kubernet's configuration utilities have the following properties: Declarative. The configuration is unmistakable, deterministic and not system-dependent.

Readable : Configuration is written in a way that is easy to understand.

Versatile: The method helps to make things easier and doesn't get in the way of doing what you're planning to do.

It's stable: The software would encourage re-use and composability.

A few key reasons Kubernet's configuration management is so challenging: what looks like a simple act of installing an application can have very specific, often overlapping, specifications, and it's impossible for a single system to satisfy all of those requirements. Imagine the following usage cases: a cluster manager who deploys third-party, off - the-shelf software, such as WordPress, to their cluster with little or no modification of those devices.

The most important criteria for this user is to easily receive upstream updates and upgrade their application as easily and seamlessly as possible (e.g. new versions, security patches, etc.).

A SaaS application developer that deploys its customized application to one or more environments (e.g. dev, staging, prod-west, prod-east). These environments can be spread across different accounts, clusters, and namespaces with subtle differences between them,so re-use configuration is paramount. For these users, it is important to move from a Git commit in their code base to a fully automated deployment in each of their environments, and to manage the configuration of their environments in a simple and manageable manner.

These developers have no interest in semantic versioning of their releases since they may be deployed multiple times a day, and the notion of major, minor and patch versions ultimately has no meaning for their application.

As you can see, these are completely different usage cases, and more often than not, a device that excels one does not treat the other very well. Since developing first-class Argo CD support for some of the more common configuration tools (Helm, kustomize, ksonnet, jsonnet) and using these tools at Intuit to handle various applications in our clusters, we've gained some unique insights into the strengths and weaknesses of each of them.

Helm

Let's start with the obvious one, Helm, that needs no introduction. Love it or hate it, Helm, as the first on the scene, is an integral part of the Kubernet community, and it is possible that you have downloaded something at one stage or another by running helm.

It is important to note that Helm is a self-described package manager for Kubernetes and does not pretend to be a configuration management system. However, since many people use Helm templates for this very purpose, it is part of the discussion. Such clients would invariably end up having multiple values.yaml, one for each setting (e.g. values-base.yaml, values-prod.yaml, values-dev.yaml), and then parameterize their list in such a manner that the values of the context can be used in the graph. This method works more or less, but it makes the templates unwieldy, as the golang template is flat, and it needs to support every conceivable parameter for each environment that ultimately litters the entire template with{-if / other} switches.

The Good: Undoubtedly, Helm's greatest strength is its excellent repository of charts. Only recently, we had a need to run a highly available Redis, with no permanent size, to be used as a throwaway cache. There's something to say about being able to throw the redis-ha chart into your namespace, set persistentVolume.enabled: false, point your service to it, and someone else has already worked hard to figure out how to run Redis reliably on a Kubernet cluster.

The Bad: a Golang model. "Look at that beautiful and elegant template for the helm!"No one has ever said that. It is well established that the Helm models have a readability issue. I don't imagine that this is going to be addressed with the help of Helm 3 for Lua, but until then, okay, I hope you enjoy curly braces.

Complicated SaaS CD tubes. In the case of SaaS CI / CD pipelines, given that you are using Helm as planned (i.e. using Tiller), automatic implementation in your pipeline can follow many routes. In the best case, distributing from your pipeline will be as easy as: docker move mycompany / guestbook: v2 helm update guestbook— set guestbook.image.tag=v2 But in the worst case, where current map parameters can not accommodate the desired manifest adjustments, you're going through the whole song and dance of bundling a new Helm graph, bumping its semvers, uploading it to a chart server, and redeploying it.

In the Linux universe, this is equivalent to building a new RPM, uploading a RPM to an yum repository, and running an yum download, all in order to get your shiny new CLI into / usr / bin. While this model works well for packaging and delivery, in the case of SaaS-based applications, it is an excessive challenge and a roundabout way to deliver the software. For this reason, many people choose to run the helm template and use the kubectl output pipe, but at that point you're better off using some other tool that's specifically designed for this purpose.

By default, non-declarative. If you have ever added— set param= value to any of your Helm installations, your deployment phase is not declarative. Such values are only documented in the Helm ConfigMap of the Netherworld (and maybe your bash history), so let's assume you've written them down somewhere.

This is far from perfect if you ever need to re-create the cluster from scratch. A slightly better option would be to log all the parameters in a new custom value.yaml that you can store in Git and execute using-f my-values.yaml. Nevertheless, this is inconvenient when you're installing an OTS map from Helm safe, and you don't have a simple place to store certain values.yaml side-by-side to the actual graph. The best solution I've come up with is to create a new dummy map that has the upstream graph as a reference. Also, I have yet to find a traditional way to change a value.yaml variable in a pipeline using a single liner, short of running sed.

Kustomize

Kustomize was built on the basis of the design principles illustrated in Brian Grant's excellent thesis on Declarative Program Management. Kustomize has seen a meteoric rise in popularity, and in the eight months since it launched, it has been integrated into a cubectl. Whether or not you agree with the way it was combined, it goes without saying that customizing apps will now have a permanent mainstay in the Kubernetes community and will be the default choice which users would gravitate towards config management. Sure, it helps to be member of the cubectl!

The Good: No constraints or models. Customize apps are extremely easy to reason for, and I dare say, a pleasure to watch. It's about as near as you can get to Kubernetes YAML because the overlays you write to render customizations are actually subsets of Kubernetes YAML. Bad: no parameters & models. The same property that allows customizing software so readable can also make it very tight.

Jsonnet

Jsonnet is simply a language and not really a "device." However, its use is not unique to Kubernetes (although it has been popularized by Kubernetes). The best way to think of jsonnet is like a super-powered JSON combined with a healthy way to template. Jsonnet incorporates all the

things you wish you could do with JSON (comments, text frames, conditions, variables, conditionals, import files), without any of the things you hate with golang / Jinja2 templates, and introduces functionality that you didn't even know you needed or wanted (functions, object orientation, mixins). This is all done in a declarative and hermetic (code as data) manner.

Jsonnet is not widely used in the Kubernetes community, which is unfortunate, because of all the tools described here, jsonnet is the hands down of the most powerful configuration tools available and is why several offshoot tools are built on top of it. More about that later. Explaining what is feasible with Jsonnet is a post in and of itself, which is why I urge you to read how Databricks utilizes Jsonnet for Kubernet, and Jsonnet's excellent learning guide.

The Good: It's extremely powerful. It's unusual to have a condition that couldn't be conveyed in a succinct and elegant snippet of jsonnet. With jsonnet, you're constantly finding new ways to maximize re-use and avoid repeating yourself.

The Bad: this isn't YAML. This might just be an unfamiliarity problem, but most people would feel a level of cognitive load when they look at a non-trivial jsonnet folder. Much as you would need to run the helm model to test that your helm map is delivering what you intend, you will also need to run jsonnet— yaml-stream guestbook.jsonnet to confirm that your jsonnet is right. The good news is that, unlike golang templates that can produce syntactically incorrect YAML due to some misplaced whitespace, jsonnet captures these types of bugs during the build process and guarantees that the resulting output is JSON / YAML valid.

Ksonnet (and other derivatives of jsonnet)

Ksonnet (wordplay in the language on which it is based) was meant to be the "Kubernet jsonnet." It provided an opinion-based way to organize your jsonnet manifests into files & directories of "components" and "environments" supported by a CLI to help facilitate the management of these files. Ksonnet made a big splash almost two years ago when Heptio and Bitnami were jointly announced, working in conjunction with Microsoft and Box, the real who is the Kubernet ecosystem.

Now, what's happened? Simply put, it was too complicated to use. When you started with ksonnet, you actually learned three things at the same time:

1. The dialect of the jsonnet itself.
2. Over-engineered concepts of ksonnet (components, prototypes, environments, parts, registries, modules).
3. Ksonnet-lib, ksonnet's jsonnet software. And if you're new to Kubernetes (as our Dev teams have been), make four of them. Argo CD started with initial support for ksonnet, and as someone who pushed for the adoption of ksonnet at Intuit, I'm sorry to see it go. Notwithstanding my own efforts to make ksonnet simpler for groups, I experienced first-hand the continuing challenges that consumers have encountered with the platform.

Including ksonnet, there are a fair number of other applications extracted from jsonnet. These include cubecfg, kapitan, kasane, kr8.

Replicated Ship is fairly new to the scene and focuses primarily on the topic of "last-mile configuration" for third-party applications. This functions by using Helm and Kustomize to create manifests. Why would you have to do that, you ask? Sooner or later, you will find yourself in a situation where the Helm Chart is almost what you want, but you still need to tweak it somehow (e.g. add network policies, set the Pod Affinity, etc...).

After studying the built-in map criteria, you understand that the graph does not provide a choice for what you're trying to do. At this stage, people usually do one of two things: apply a PR upstream to attach another variable to the map, or dump the contents of the helm model into the folder and hand-edit the YAML with the desired changes.

The trouble with the latter is that it can't be maintained. When it's time to get the latest from upstream, you can't easily see or reapply the changes you've made to your fork. Ship solves this by keeping the following separate: the tracking reference and the staging directory hold the upstream helm chart, along with your last-mile modifications written as customization overlays. Using this method, the standard manifests you obtain from upstream can be changed independently of your local customizations.

The Good : It's filling the gap. Ship makes it easy for operators to configure a map without having to push upstream. There's even a great UI to help guide you through this process.

The Bad: Functions with OTS only. Ship is only intended to be used with an off - the-shelf, upstream source, which means that it does not handle a case-by-case configuration of the application.

We're not going to need a plane. I find it unfortunate that we even need a ship-like tool. This is not a problem with Ship itself, but a commentary on the weaknesses of our current tools. Imagine, for instance, if Helm had an automated way to add basic overlays to upstream graphs (as an alternative to parameters), we would end up with much clearer charts without the chaos of excessively parameterized models. Another example is to imagine a world where everyone has personalized software for their works. If this was prevalent, users could use the remote base feature of Customize to apply local changes to upstream customization apps. But unless either of these things happens, Ship is a resource to bridge that

gap.

Helm 3 and Lua Script

Unfortunately, this aspect of Helm 3 is one of the least developed, and the maintainers have only just started writing the underlying Lua VM that will power the rendering engine, so it will be quite some time before we have more readable charts. I assume they will fix the last-mile configuration issue and be more accommodating to the GitOps deployment model with the Helm 3 update.

The Kubernet configuration management is at an inflection point. Customize is now readily available at the disposal of the consumer, making it easier than ever to direct consumers with extremely capable out - of-the-box configuration management. But if there's a single lesson out of this debate, it's because there's no flawless configuration management method, so they tend to come and go as the wind blows. Every method will have its strengths and weaknesses, so it's important to understand when to use the right tool for the job.

Right now, we've got a mix of apps defined in Customize, Helm, Ksonnet, all in the same cluster for different reasons. For Argo CD, the guiding principle was to provide clients with the most versatility in this respect, such as the ability to configure the repo database and the ability to execute specific commands to produce manifests.

The state of Kubernetes config management has never been more exciting (as exciting as writing more textcan be:-). The merge of Kustomize into cubectl completely changes the game. The departure of Ksonnet from the market is a symbol of the ripening of storage. Jsonnet is always going to have a place for power users. And while Helm has lost some sense of community late with Tiller's security concerns and design complexities, he's got a wildcard up his sleeve with Helm 3 and the new Lua graphs.

At the very least, Kubernet's configuration management is a fast-moving environment that involves all Kubernet users in some way or another, and everyone should have a vested interest in how things will be formed in the years to come. Until we have a clear winner, please keep using the Argo CD with the device of your choice!

The New Basics of Configuration Management in Kubernetes

In the context of mainstream cloud technologies, enterprise IT migration to containerization in general and to Kubernets in particular is well underway. Several companies are shifting in reaction to competitive pressures and the need for greater business flexibility. Others are making the switch for economic reasons; they want more cost-effective IT services, and they see Kubernetes as a smart way to get there.

The momentum of this push to Kubernetes is understandable. The strengths are too important to dismiss. For IT services, frameworks are more versatile and flexible than substitutes, easier to develop and easier to deploy, faster and cheaper. Essentially, Kubernetes allows businesses to help their growth and change in a nimble, productive and cost-effective manner.

This is the word. But the reality is that DevOps and IT teams in many organizations still can't quite get their Kubernet-powered operations to "fly right." That's because of the complexity of the system. This is partly due to the versatility of Kubernetes, which offers the developers seemingly endless options and choices. However, this flexibility is becoming more complex as the teams initially work to get their clusters up and running. For clusters up but programs not running to their taste, groups instead seek to change their software. That's when Kubernets first hit the wall of uncertainty.

In companies that are early in their Kubernet trip, this difficulty makes it difficult in their teams to get apps to be delivered reliably and consistently high-performance. For enterprises that are further along in their Kubernet migrations, complexity is what prevents them from achieving their expected cost savings.

Old-School Approach Falls Short As for software products that help teams overcome the complexity of their Kubernets, the options have been limited. There is no shortage of Kubernet cluster deployment services and application performance monitoring products. To date, however, there have been no solutions specifically designed to optimize how applications run in Kubernetes environments.

Despite software-driven solutions, DevOps and IT managers have struggled with it in an old fashion— manually through trial and error. They adjust one or two factors, then they nervously wait to see the effect. It's often unclear why changing "A" caused "B" to break, so they keep on tinkering. In organizations where software quality is key, such as SaaS providers or MSPs, their teams sometimes rely on expensive over-provisioning.

Configuration management in Kubernetes is a multidimensional chess game which DevOps and IT managers are losing too often.

Consequently, the complexity-related problems referred to above. Some of these arise at the cluster stage, such as having to decide how big to render nodes and how many of them to build. But there are many more problems at the application level.

Let's look at a web app like an e-commerce site as an example. Minimizing latency is crucial to a smooth user experience, so this is a key consideration. For order to achieve that aim reliably, the device has to be finely designed.

When the app is deployed in Kubernetes, it is up to DevOps or an IT team member to select a number of instances and choose how much CPU,

memory, and other types of resources to allocate to each instance. Allocate too few resources, and the device can slow down or even crash. Allocate too many money, and all of a sudden the server costs spike. It's a big order to figure out the "just right" configuration settings, and do so quickly, accurately, and consistently for a growing list of apps.

The fact is that configuration management in Kubernetes is a multidimensional chess game, and one that DevOps and IT teams are losing too often. To succeed, and to do so successfully, they need a better way forward.

A Smarter, More Effective Way Forward Emerges There is good news for DevOps and IT teams that are currently struggling with the complexity of Kubernetes. A modern, software-driven approach to the fundamentals of system design in Kubernet environments has arisen. Powered by sophisticated machine learning, this new approach reduces most of this ambiguity by automatically defining optimum system configuration parameters.

These technologies, which build on established data science methods, allow DevOps teams to automate the parameter tuning process, freeing them to focus on other mission-critical tasks. Using machine learning-powered testing, these solutions allow productive exploration of the system parameter area, resulting in configurations that are assured to be both stable and optimized.

As with all powerful ML techniques, the ability to learn over time plays a key role in making the process scalable and more efficient. With the aid of these tools, teams will rest assured that the design and size of their projects can automatically flow into the automation system, which will become increasingly sophisticated with time.

In short, ML-powered methods to installing, integrating, scaling and handling containerized software in Kubernet's ecosystems are coming to the fore. We show themselves by intelligently evaluating and handling hundreds of interrelated parameters with millions of potential variations to automatically pick the best settings for each program.

Using our web app example, rather the DevOps team struggling to determine the best parameter values for their device, with these latest fundamentals of config optimization, the team can instantly provide customized parameters to them. In addition, both the organization and its customers benefit from a more reliable, high-quality user experience.

It's all about high performance and cost-effective reliability. Through allowing faster and more efficient delivery of applications and ensuring that they are properly resourced and optimally optimized, a modern, ML-based approach will be a trigger for even more Kubernet acceptance and performance. And that's a really good thing.

Kubernetes Helm is a map management tool. Charts are pre-configured in Kubernet resource packages.

Discover and use popular software bundled as Helm Charts to operate in Kubernetes Upload your own applications as Helm Charts Build reproducible versions of your Kubernetes applications Intelligently maintain your Kubernetes manifest files Maintain Helm product updates in Helm Handbasket Helm is a resource that streamlines downloading and handling Kubernetes applications. Thought of it as apt / yum / homebrew for Kubernetes.

Helm makes the models and connects with the Kubernetes Server Helm running on your desktop, CI / CD, or wherever you want it to run.

Charts are Helm modules which contain at least two things: a product summary (Chart.yaml) One or more models which include Kubernet manifest files Charts can be placed on a server or downloaded from external chart servers (such as Debian or RedHat packs) File versions of the Helm software can be located on the Releases site.

Unpack the differential helm and link it to the PATH, and you're free to go!

If you want to use a package manager: homebrew users can use brew to install kubernetes-helm.

Chocolatey users can use Chocolate to install Kubernetes-helm.

Scoop clients can use the scoop to download the helm.

GoFish users can use gofish to download the helm.

Start with the Quick Start Guide to get Helm up and running quickly.

_____

Docs Get going with the Quick Start Guide or immerse yourself in the full documentation Roadmap The Helm Roadmap uses Github indicators to track the progress of the venture.

Community, discussion, contribution and support You can reach the Helm community and developers via the following channels: Kubernetes Slack: #helm-users #helm-dev #charts Mailing List: Helm Mailing List Developer Call: Thursdays 9:30-10:00 Pacific. https:/zoom.us/j/696660622 Code of

conduct Membership in the Helm Society shall be regulated by the Code of Conduct.

Introduction to Helm, Package Manager for Kubernets Introduction Applications for Kubernets–a powerful and popular container-orchestration framework–can be complicated. Setting up a single application can involve creating several interdependent Kubernet assets–such as pods, modules, deployments, and replicasets–each requiring you to write a detailed YAML manifest document.

Helm is a Kubernet package manager that allows developers and operators to package, configure and deploy applications and services more easily to Kubernetes clusters.

Helm is now an official Kubernetes project and is part of the Cloud Native Computing Foundation, a non-profit organization that supports open source projects in and around the Kubernetes ecosystem.

Helm Overview

Most programming languages and operating systems have their own package manager to help with the installation and maintenance of software. Helm provides the same basic feature set as many of the package managers that you may already be familiar with, such as Debian's apt or Python's pip.

Helm could:

Install the software.

Download code modules automatically.

Upgrade the code.

Set up software deployments.

Choose software packages from servers.

Helm delivers this capability through the following components: the command line tool, the helm, which offers the user interface for all Helm applications.

Companion server module, tiller, which runs on your Kubernet cluster, listens to commands from the helm, and handles the setup and delivery of software releases on the cluster.

The format of the Helm packaging, called charts.

Official curated chart repository with prepackaged charts for popular open-source software projects.

Next, we'll look at the charts format in more detail.

Charts Helm packages are called charts and consist of a few YAML configuration files and some templates that are rendered into Kubernet manifest files. Here is the simple folder structure of the chart: Example chart directory package-name / charts / templates / Chart.yaml LICENSE README.md requirements.yaml values.yaml Such folders and files have the following functions:charts/: Manually controlled chart dependencies can be located in this directory, although it is usually better to use requirements.yaml dynamically related.

Templates/: This folder includes template files that are paired with configuration values (from values.yaml and command line) and made into Kubernet manifests. The designs use the Go programming language design format.

Chart.yaml: A YAML file with chart metadata, such as chart name and version, maintainer information, relevant website, and keyword search.

Authorization: a plain-text authorization for the map.

README.md: A readme file with chart user information.

Requirements.yaml: A YAML file that lists the dependencies of the chart.

Values.yaml: A YAML folder with the standard configuration values for the map.

The helm command may load a map from a local folder or from a.tar.gz bundled copy of this file structure. These packaged charts can also be downloaded and installed automatically from repositories of charts or repositories.

Map Repositories : A Helm Graph Repo is a basic HTTP page that contains an index.yaml directory and a.tar.gz packed map. The helm command has subcommands available to help bundle charts and to construct the appropriate index.yaml folder. Every web server, object storage system, or static page host, such as GitHub Pages, may handle such documents.

Helm comes pre-configured with the standard map database, referred to as secure. This archive leads to the Google Storage pool at https:/kubernetes-charts.storage.googleapis.com. The reference for the secure directory can be found in GitHub's helm / chart folder.

Alternate repos can be added with the command helm repo add. Some popular alternative repositories are: the official incubator repo containing charts that are not yet ready for stable repositories. Instructions for using the incubator can be found on the GitHub Official Helm Charts site.

Bitnami Helm Graphs which provide certain graphs that are not included in the official stable repo.

Whether you're installing a chart that you've developed locally or a chart from a repo, you're going to need to configure it for your specific setup.

Graph Setup : A graph usually comes with default setup values in a.yaml folder. Many programs may be entirely deployable with default values, but you will usually need to change some of the specification to meet your needs.

The values that are shown for configuration are calculated by the writer of the map. Most are used to customize basic Kubernets, and some can be transferred to the underlying container to configure the request itself.

Here is a snapshot of some of the following sample values: values.yaml service: type: ClusterIP port:3306 These are the choices for configuring the Kubernetes Service tool. You can use the helm inspect chart-name values to dump all of the configuration values available for the chart.

Such values can be overridden by creating your own YAML folder and using it while loading the helm, or by selecting the options individually on the — set flag command line. All you need to do is decide the attributes you want to switch from the defaults.

The Helm Chart that is distributed with a particular configuration is called a launch. Next, we'll talk about releases.

Releases

When a map is mounted, Helm combines the graph models with the configuration defined by the client and the defaults in value.yaml. These are turned into Kubernet manifests that are then implemented via the Kubernetes API. It allows the creation, setup and implementation of a particular map.

The release principle is essential because you may want to run the same program more than once on a cluster. Of example, you may need several MySQL servers with different configurations.

You're often going to want to update different instances of a map individually. Every program may be equipped for an upgraded MySQL database, but another is not. For Helm, you can update each launch individually.

You may change the release because the map has been changed, or because you want to modify the setup of the release. Any way, that update will create a new version of the release, and Helm would allow you to roll back to previous versions quickly if there is an issue.

Create Charts When you can't find an established map for the technology you are installing, you may want to build your own graph. Helm can output a chart-name scaffold in the chart directory. This will create a folder containing the files and directories we discussed in the Charts section above.

From there, you'll want to fill out your map metadata in Chart.yaml and place your Kubernet manifest files in the templates tab. You will then need to remove the appropriate configuration variables from your manifests and into values.yaml, and then add them to your manifest models using the model method.

The helm control has a range of subcommands available to help you check, organize, and represent your maps. Please read the official Helm documentation on the development of charts for more information.

CHAPTER TEN

Kubernetes Helm

Helm has two sections , the client(CLI) and the server(Tiller). The client lives on your local workstation, and the database lives on the kubernet cluster to execute what is required. The premise is that you use the CLI to move the services you need, and the tiller must make sure that the condition is actually the case by creating / updating / delete assets from the graph. To fully grasp the helm, there are three concepts that we need to familiarize ourselves with: Chart: a package of pre-configured Kubernet resources.

Release: a particular instance of a map that has been distributed to a cluster utilizing Helm.

Repository: A group of published charts that can be made available to others.

Benefit of Using Helm

Less Duplication: This is obviously the obvious one provided once the map has been created, it can be used over and over again and by anyone else.

Less Complexity : The fact that you can use the same graph for any environment reduces the complexity of creating something for dev, test and prod. You can just change the map to make sure it's ready to apply to any setting. And you get the advantage of using a ready-made map in Dev.

Less Learning Curve : this is another factor. K8s is hard to grasp if you're new to it, so what helm offers us is the advantage of encouraging every designer to concentrate on app development as opposed to learning a new platform. All you need to do is identify what you need to customize. Again from the first example, less duplicity implies that you can take an established map and use it for your use without any problems.

The Map Helm definition has a certain form when you create a new map. Run "Print YOUR-CHART-NAME" to print. When this has been developed, the folder structure will look like:

Copy

YOUR-CHART-NAME/

|

|- .helmignore

|

|- Chart.yaml

|

|- values.yaml

|

|- charts/

|

|- templates/

.helmignore: This holds all the files to be ignored when the chart is packed. Close to.gitignore if you're comfortable with git.

Chart.yaml: This is where you put all the information on the chart you are packing. So, for example, your version number, etc. This is where you're going to put all the info.

Values.yaml: This is where you describe all the values that you want to add into your models. If you're acquainted with terraform, find this folder as variable.tf helms.

Graphs: This is where you place the other graphs on which your map rests. You might be calling another chart that your chart needs to work properly.

Templates: This file is where you placed the real blueprint that you are adding to the map. For instance, you might be deploying a nginx installation which requires a network, a configmap, and a password. You're going to have your deployment.yaml, service.yaml, config.yaml, and secrets.yaml in the design folder. They're both going to get their beliefs from the values.yaml above.

Installing Helm This is a very simple process. Below, you can download the binary version. Unpack the binary helm and add it to your PATH, and that's all you need. You can also download this with a package manager or homebrew. Once installed, you can simply run "helm init" to install the tiller server in your k8s cluster.

Helm is ideal if you're searching for a structured way to send your software to the Kubernets. It's very useful to help you concentrate more on the actual development as opposed to learning the ins and outs of kubernets. Take a look at this project.

CHAPTER ELEVEN

The Meaning Of Terms

Cloud-Native

Cloud-native is a term used to describe container-based systems. Cloud-native architectures are used to develop applications developed with

container-packaged resources, distributed as microservices and operated on an open network via agile DevOps processes and continuous delivery. It is a two-fold concept and a form of approach to building applications and services specifically for the cloud environment. These are also the features of those applications and services.

Microservice

This is a framework of structuring and designing software. Rather than having a large, monolithic system where all the functions are interdependent, the software is broken down into a series of smaller components (conveniently referred to as microservices).

Microservice may be implemented independently of the remainder of the request. For example, an online retailer could have one microservice for inventory management and another for managing their shipping services. The key point to remember is that before you can take advantage of Kubernetes, your applications need a microservice architecture.

_____

Desired State

The desired state is one of the core concepts of Kubernet. A declarative or imperative API is used to describe the state of the entities (Pod, ReplicatSet, Deployment etc.) that will operate the containers.

Containers

A container is a structured framework that can hold and operate any program. Initially, containers are merely considered an alternative for Virtual Machines (VMs) when it comes to virtualization technologies, but now they are also the "go-to" of virtualization.

Then you re-architected your software to the microservices. -microservice can now be housed in its own container, along with everything it needs to run (settings, libraries, dependencies, etc.). This creates an isolated operating environment so that the container can run on any machine.

Docker

If you've learned of the boxes, you're almost probably going to come across Docker, too. Docker is the industry's leading containerization software, currently with a market share of about 95%.

Pod

A pod contains one or more bottles. Pods are the smallest category inside Kubernetes (this is why containers are theoretically not part of Kubernetes, as even a single container is considered a pod).

All containers in the pod share infrastructure and network and can connect with each other, even if they are on different nodes.

The hardware components are nodes. A node is likely to be a virtual machine hosted by a cloud provider or a physical machine in a data center. But it can be easier to think about nodes as CPU / RAM assets to be used by the Kubernet cluster than as individual computers. This is because pods are not limited to any given machine at any given time, they will move across all available resources to achieve the desired status of the application.

There are two forms of nodes–the worker and the leader.

_____

Cluster

Clusters actually run the containerized software that Kubernetes handles. A cluster is a set of nodes that are linked together.

By joining together, the nodes pool their resources, making the cluster much more powerful than the individual machines it makes up. Kubernetes moves pods around the cluster as nodes are added or removed.

A cluster consists of multiple worker nodes and at least one master node.

Services

A system is an API entity which provides a request, which basically defines how network traffic can reach a collection of pods. Resources can be located on every node.

Deployment

Deployments are an Interface entity that handles pod replication.

Deployment determines the state of your group–for instance, how many replicas of a pod should be operating. When the deployment is connected to the cluster, Kubernets must immediately make the correct number of pods and then track them. If the pod crashes, Kubernetes must duplicate it according to the' deployment' requirements.

_____

Kubeadm

Kubeadm is a quick -start development device for Kubernets. It lets you build a minimum viable cluster with a single master node. Kubeadm is quick and easy to use. Besides, it means that your cluster complies with best practices, so it's a great tool to use when reviewing Kubernets for the first time or when checking the applications.

Minikube

Minikube is a lightweight version of Kubernetes that is much easier to use locally. It will build a VM on your local machine where you can operate a single node cluster. This is very useful for testing.

_____

Worker Node

Worker nodes are the locations where pods are installed. Kubernetes has a master-slave model, so a worker's node is a slave.

Kubelet

Kubelet is an agent that operates on every worker's node in a cluster. This is an important component, as it derives orders from the master node.

By fact, the kubelet controls the cells. This means that all containers operate in a pod and that these pods are stable and operated at the right point in time. It will then release and destroy pods as it provides guidance from the master node as to which pods need to be inserted or deleted.

Kube-proxy

Kube-proxy is a proxy for the Kubernet network. It's a service that runs on every node and handles forwarding requests.

The main job of kube-proxy is to set up rules for iptables. It deals with the fact that different pods will have different IP addresses and enable you to link to any pod inside Kubernets, which is vital for allowing load balancing.

If you're searching for a little of technical detail, it's UDP, TCP, and SCTP, but you don't get HTTP.

Master Node

Master Nodes monitors the installation of the containers, and therefore the employee nodes.

As the name suggests, these nodes are the rulers of the Kubernet master-slave model.

Etcd

Etcd stores configuration information for large, distributed systems. Thus, Kubernetes uses etcd as a key value store for configuration management and application exploration. Etcd must be readily accessible and reliable in order to ensure that facilities are correctly planned and run. Because information from etcd is so important, it is highly recommended that you have a backup for your cluster.

Scheduler (kube-scheduler)

The scheduler is a part of the master node and does what it says on the tin–takes scheduling choices for newly created pods. When a pod is created, a node must be assigned to run on it.

The scheduler shall receive information from the API on the requirements and specifications of the pods and the IT resources of the available nodes. Then each pod will be assigned to the appropriate node. If the scheduler can not find an appropriate node, the pod will stay unscheduled and the scheduler will restore the cycle until the node is accessible.

_____

Controller

A controller is an element on the master node, it acts to transfer the program from its current state to the desired state. Kubernetes has a wide

range of controls. For example, the Replication Controller guarantees that the correct number of pod replicas are operating at any given time.

The controller must' watch' the API for data on the cluster state and make changes accordingly. So, if the API specified that there were 7 replicas of podA, but there were currently 8, the Replication Controller would terminate the extra pod.

Kube-controller-manager

This is a master node component that manages all controllers. Every unit, for example, Replication Controller, Namespac eController etc. is a different step or process. However, in order to simplify cluster management, they are all compiled into a single process. The kube-controller-manager will be responsible for this recording.

It's also a way to add unique cloud provider capabilities to clusters. It helps to keep the vendor-agnostic Kubernetes.

The kube-apiserver

Kube-apiserver is a master node component that reveals the Kubernetes API to the master. It essentially acts as a frontend for data on the collective state of the cluster, offering a connection between the API and other Kubernet artifacts, e.g. pods, controllers, utilities, etc. So all communications between the components go through the kube-apiserver.

Kubernets API

The Kubernets API allows Kubernets to run. This holds data about the state of the group. Data on the desired state of the cluster is inserted into the API via the user interface or the command line interface. The Kubernetes API then forwards this information to the master node.

The master node will then order the worker nodes to carry out behavior that will move the program from its current state to the desired state.

Kubectl

Kubectl is the latest Kubernet command line interface tool. It's used to connect with the API.

You can use the kubectl command to build, examine, modify, duplicate and remove Kubernet artifacts, e.g. pods and nodes.

Aggregation Layer The aggregation layer allows you to extend Kubernets by installing additional APIs. This can have more flexibility than just using the Kubernetes Framework base. For instance, you can incorporate API extensions that enable you to create new controllers.

Name

Names are special identifiers of Kubernet entities issued by the user. At any given time, only one object can have a specific name. But, if this entity is removed, you will create a new item using the same name.

Names apply to a property URL link,e.g./api /v2/pods / test-name. These can be up to 253 characters and may contain lowercase alphanumeric characters,." "and-."

• UIDs

UIDs are unique identifiers for Kubernetes objects, they are systems-generated. Unlike with names, every object created over the lifetime of a Kubernetes cluster has its own UID. So, even if you delete an object you can’t then reuse the same UID when you create a new object.

Here is an example of a UID: 946d785e-998a-12e7-a8dd-42010a800007, because they are systems-generated they are typically less user-friendly than names. But, the fact that they are only ever used once makes UIDs useful for distinguishing between historical instances of similar objects.

• Annotations

Annotations are key-value pairs which are used to attach non-identifying metadata to Kubernetes objects.

• Labels

Labels are key-value pairs which are used to attach meaningful, identifying metadata to Kubernetes objects.

Selector

Selectors are used for querying Kubernetes resources and filter them by their labels. A selector allows you to identify a set of objects with shared characteristic(s).

There are equality-based selectors, which allow you to identify all resources that possess a certain label. For example, “environment = dev” would filter a list of all resources that are running in a dev environment.

There are also set-based selectors, which allow you to identify all resources that match a set of values. For example, “environment in (production) tier notin ” would provide a list of all resources in a production environment, excluding any labelled as backend.

- Pod Preset

PodPreset is an API object that automates the addition of information to pods as they are created.

The PodPreset uses selectors to determine which pods should receive which information. You can add information such environment variables, secrets (service account ‘credentials’) and volumes using PodPreset.

This object saves time, as pod template authors don’t have to provide all information for each pod. It also has benefits for access control, as it means authors also don’t need to know all of the details about specific services on a pod.

- Role-Based Access Control (RBAC)

RBAC allows Kubernetes admins to configure access policies through the API. It allows you to assign different roles to users which gives them varying permissions.

If you’re familiar with WordPress blogs, RBAC works in a similar way. The functions you can perform within Kubernetes depends on the permissions level your role allows. Just like how in WordPress an Author can post new articles, but can’t update existing ones. Whereas an Editor both publish new articles and edit existing ones.

Service Account

Service accounts are used to identify processes that run in a pod. All pods have a service account, if you do not set a specific service account for a pod then it will be set to default.

Service accounts are similar to a user account that you have to log into a website. You can only access the website if your user account has the necessary credentials to successfully log in. In the same way, a process in a pod can only run if it is authenticated by the API to access cluster resources. The Kubernetes API will only authenticate the process if the service account provides the necessary credentials.

- Container Environment Variable

Container environment variables are name-value pairs which provide meaningful information to the containers running in a pod.

This information can be about the container itself or about resources which are important for running the containerised application. An example would be details of the pod IP address or file system details.

- Taint

A taint is a key-value pair which prevents pods from being scheduled on nodes/node groups. Taints are specified on nodes

- Toleration

A toleration is a key-value pair specified on pods which enables pods to be scheduled on nodes/nodegroups. Tolerations are specified on pods.

Taints and tolerations work together, the goal is to ensure that pods are only scheduled on appropriate nodes. If the toleration on the pod matches the taint on the node then the pod is allowed to be scheduled on that node. But, if the toleration on the pod does not match the taint on the node, then the node will not accept the pod.

- Namespace

Namespaces are an abstraction used in Kubernetes. It allows you to have multiple virtual clusters within the same physical cluster. Each of these virtual clusters is a namespace.

Namespaces are only really useful if there are many users working with your Kubernetes cluster across multiple teams and projects. They can help you to organise and divide cluster resources. One main benefit is around names, object names only need to be unique within a namespace. So you can use the same name for objects across different namespaces.

- Job

A job is a task that runs to completion.The task is classed as complete when the required number of pods have completed and terminated. The job tracks the successful completions of the pods.

The simplest jobs run one pod to completion, the job will start a new pod if the original fails for any reason. But a job can also be used to run

multiple pods in parallel.

- Cronjob

Cronjob manages jobs which run on a schedule, it also allows you to add some specific parameters for how job failure should be handled.

- StatefulSets

Stateless applications generally make Kubernetes a lot easier, but there can be issues with data loss if a pod fails. StatefulSets helps to overcome this.

With StatefulSets, you can assign a pod a number, and assign resources to that number – such as volumes, network IDs and other indexes. So if a pod fails, it'll be restored with the same data it had previously.

- DaemonSet

A DaemonSet is used to deploy system daemons that power Kubernetes and the OS e.g. log collectors or node monitoring agents.

These system daemons are typically required on every node, so the DaemonSet ensures that a copy of these daemon pods is running on every node. When a new node is spun up, so is a new copy of the DaemonSet's pods.

- Probes

Probes check the liveness and readiness of containers within pods.

If a liveness probe detects an issue where an application is running but not making progress then it can restart a container to increase the applications' availability.

Readiness probes check when a container is ready to start accepting traffic. A pod is ready once all of its containers are ready.

These probes enable exciting DevOps improvements – such as zero downtime deploys, preventing broken deploys and self-recovering containers.

- Operators

Operators are deployed to a Kubernetes cluster and enable you to write an application to fully manage another.

An operator can monitor, change pods/services, scale the cluster up/down and even call endpoints in running applications. For example, an operator could detect if a pod is running at above 90% CPU and provision more resource automatically to keep it running.

- Persistent Volumes

Persistent Volumes (PVs) are used for static provisioning, where storage has been created in advance.

So, when a new pod is created the author can request a PV which will reference an existing piece of storage space in the cluster. This storage space is then assigned to the pod. PVs persist beyond the lifecycle of individual pods. So when a pod is deleted, the storage it was using (and the PV that represents that storage) still exist.

PVs are useful because objects that run on your cluster will not run on the same node all the time, so they need somewhere to store data.

- Storage Class

Storage classes are used to define the different storage types available on the cluster.

You can define various properties including the provisioner, volume parameters and reclaim policy. Storage classes are used alongside persistent volumes (PVs) so that users can access PVs with the correct properties for their storage needs, without being exposed to the explicit details of how those volumes are implemented.

- Network Policy

The network policy specifies how pods can communicate with each other and with other network endpoints e.g. ports.

By default, all pods are non-isolated meaning they can connect to any other pod and accept traffic from any source. But, you can use the network policy to stop certain pods from connecting to each other and to reject traffic from certain ports. The network policy uses labels to group the pods and define the specified rules for each group.

- Ingress

Ingress is the ‘door’ to your Kubernetes cluster. It is an API object that allows external traffic to access the services in your cluster.

For example, you could have a web ingress allowing port 80 traffic to your application. As it is responsible for external traffic, Ingress can provide load balancing.

- Custom Resource Definition

A custom resource definition (CRD) allows you to extend the Kubernetes API beyond the built-in resources without having to build a full custom server.

So by default in the Kubernetes API, there is a built-in pods resource which stores all of the pods in your cluster. By using a CRD, you can create a new resource to store custom objects which you have defined.

To create a CRD you simply create a YAML file containing the necessary fields. When you deploy the CRD file into the cluster, the API will then begin serving the new custom object.

- Resource Quota

Resource quotas are applied to namespaces to constrain their resource consumption.

Two main limits are set. Firstly, on the quantity of each type of object e.g. pods or services that can be created in each namespace. Secondly, on the total compute resources that can be consumed by objects in that namespace.

- Horizontal Pod Autoscaler (HPA)

HPA is an API resource which automates pod replication.

It automatically scales the number of pod replicas running at any given time to reach CPU utilization targets or based on information from the Replication Controller about the replicas needed for the application to move towards its desired state.

Container Storage Interface (CSI)

The CSI provides a standard interface between storage systems and containers.

The CSI is what enables custom storage plugins created by other vendors to work with Kubernetes without needing to add them to the Kubernetes repository.

- Container Network Interface (CNI)

The CNI provides a standard interface between network providers and Kubernetes networking.

Kubenet is the default network provider in Kubernetes, it is generally pretty good but it is basic. The CNI allows you to use other network providers which offer more advanced features e.g. BGP and mesh networking if this is something you need for your cluster.

Helm

Helm is an application package manager which runs on top of Kubernetes. It allows you to describe your application structure using helm charts and manage the structure using simple commands. The tool allows you to clearly define each microservice within your application and only scale the necessary ones (by spinning up more pods and nodes in Kubernetes as needed).

- Tiller

Tiller is the “in-cluster” portion of Helm which interacts directly with the Kubernetes API to allow you to add, upgrade, query or delete Kubernetes resources using Helm.

Ceph

It is a distributed storage system created with scalability in mind. It is an example of a custom storage plugin that you can use with Kubernetes.

You can use Helm to deploy Ceph in a Kubernetes environment to add durable storage.

- Prometheus

Prometheus is a solution for monitoring metrics on Kubernetes. It is a great tool if you are looking to scale your Kubernetes cluster.

Prometheus runs on top of Kubernetes and provides a metrics dashboard which your team can use to proactively monitor your cluster. You can

also set it up to trigger alerts and notifications for certain metrics.

- Istio

Istio is a tool for connecting, monitoring and securing microservices. It can be deployed on Kubernetes to help your development team handle traffic management, load balancing, access control and more for your containerized application.