* What is docker compose file?

The Compose file provides a way to document and configure all of the application's service dependencies (databases, queues, caches, web service APIs, etc). Using the Compose command line tool you can create and start one or more containers for each dependency with a single command ( docker-compose up ).

* **How to reset, revert, and return to previous states in Git?**

Let's start with the Git command reset. Practically, you can think of it as a "rollback"—it points your local environment back to a previous commit. By "local environment," we mean your local repository, staging area, and working directory.

* **What is the difference between pull and clone in git?**
* git clone is how you get a local copy of an existing repository to work on. ... git pull (or git fetch + git merge ) is how you update that local copy with new commits from the remote repository
* **What is git rebase?**

Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow.

* What is artifact?

An artifact is a file, usually a JAR, that gets deployed to a Maven repository. A Maven build produces one or more artifacts, such as a compiled JAR and a "sources" JAR. Each artifact has a group ID (usually a reversed domain name, like com.example.foo), an artifact ID (just a name), and a version string. The three together uniquely identify the artifact. A project's dependencies are specified as artifacts.

* Diff Kubernetes Secrets vs ConfigMaps ?

The major difference is, Secrets store data in base64 format meanwhile ConfigMaps store data in a plain text. If you have some critical data like, keys, passwords, service accounts credentials, db connection string, etc then you should always go for Secrets rather than Configs.

* How do you stop a running job in Jenkins?

Go to "Manage Jenkins" > "Script Console" and run a script: Jenkins . instance.

If you have an unstoppable Pipeline job, try the following:

Abort the job by clicking the red X next to the build progress bar.

Click on "Pause/resume" on the build to pause.

Click on "Pause/resume" again to resume the build.

**Benefits & drawbacks of Kubernetes**

***Benefits of Kubernetes:***

* Kubernetes is backed by the Cloud Native Computing Foundation (CNCF).
* Kubernetes have an impressively huge community among container orchestration tools. Over 50,000 commits and 1200 contributors.
* Kubernetes is an open source and modular tool that works with any OS.
* Kubernetes provides easy service organization with pods

***Drawbacks of Kubernetes***

* When doing it yourself, Kubernetes installation can be quite complex with steep learning curve. An option to solve this issue is to opt for a managed [Kubernetes-as-a-service](https://vexxhost.com/public-cloud/container-services/kubernetes/) such as ours.
* In Kubernetes, it is required to have a separate set of tools for management, including kubectl CLI.
* It is Incompatible with existing Docker CLI and Compose tools

**Benefits & drawbacks of Docker Swarm**

***Benefits of Docker Swarm***

* Docker Swarm is easy to install with a fast setup
* Docker Swarm is a lightweight installation. It is simpler to deploy and Swarm mode is included in the Docker engine.
* Docker Swarm has an easier learning curve.
* Docker Swarm smoothly integrates with Docker Compose and Docker CLI. That’s because these are native Docker tools. Most of the Docker CLI commands will work with Swarm.

***Drawbacks of Docker Swarm***

* Docker Swarm provides limited functionality.
* Docker Swarm has limited fault tolerance.
* Docker Swarm have smaller community and project as compared to Kubernetes community
* In Docker Swarm, services can be scaled manually.
* What container runtime are you using for Kubernetes?

This page lists details for using several common container runtimes with Kubernetes, on Linux:

Docker

* What is diff betw Jenkins pipeline and freestyle project?

In freestyle project if something fails, you have to execute all the the stages again but when you write a Jenkins pipeline through Jenkins file, you specify all the stages in there and you can execute whichever stage you want.

* what are the components of the control plane in Kubernetes?

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine. See Building High-Availability Clusters for an example multi-master-VM setup.

**kube-apiserver**

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

**etcd**

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for those data. You can find in-depth information about etcd in the official documentation.

kube-scheduler

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

kube-controller-manager

Control Plane component that runs controller processes. Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

Node controller: Responsible for noticing and responding when nodes go down.

*Job controller*: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).

Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

***cloud-controller-manager***

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.

The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

The following controllers can have cloud provider dependencies:

Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding

Route controller: For setting up routes in the underlying cloud infrastructure

Service controller: For creating, updating and deleting cloud provider load balancers

Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

Container runtime

The container runtime is the software that is responsible for running containers.

Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface)

* **what is replica set in Kubernetes?**

Basically, Replica set is also a replication controller that can also work on 'set based' selector. RS can manage very big environments compared to RC.

* **Use Count to Provision Multiple Resources terraform**

When using count, Terraform will replicate the given resource a specified number of times. The following example will provision two similar instances using the same block of configuration.

* Example configuration using count
* You can also use a variable to define the number of resources provisioned with count, making the configuration even more flexible.
* **Use For Each For More Flexible Configuration**
* Like count, for\_each will provision multiple resources, but instead of using an integer to define the number of resources, for\_each uses a data structure, creating one copy of the given resource for each item in the data structure. This allows you to configure the individual resources in more complex ways. The following configuration will provision a VPC for each element in the map called project, each with a configurable number of public and private subnets.
* 7**. Release often**
* Frequent releases are only possible if the software is in a release-ready state and you have tested it in a production-like environment. That’s why the best practice is to add a deployment stage which closely resembles the production environment before the release. Some release best practices include:
* **Canary deployment** . Releasing to a subset of users, testing with that base and rolling it out to the wider population if successful (or rolling it back for iteration if it’s not).
* **Blue green deployment** . You begin with two identical production environments. One is live in production. The other is idle. When a new release is rolled out the changes are pushed to the idle environment. Then they switch - the environment containing the new release becomes the live environment. If something goes wrong, you can immediately roll back to the other environment (the one that does not contain the new release). If all is well - the environments are brought to parity once more.
* **A/B Testing.** A/B testing is similar in flavor to - but not to be confused with- blue green deployments. A/B testing is a way of testing features within the application for things like usability. The better performing variant of the feature wins. This is not a release methodology
* **Changing the SSH port with** Ansible – In a nutshell, you edit the Port parameter of /etc/ssh/sshd\_config , restart sshd and you're done.
* **Can you name 5 modules in Ansible?**

Ping Module

Ping is used when we want to check whether the connection with our hosts defined in the inventory file is established or not.

ansible test-servers -m ping -u ec2-user

ping changes to pong if an SSH connection is established.

Setup Module

The setup module is used when we want to see the information of all the hosts, their configuration, and detailed information.

ansible test-servers -m setup -u ec2-user

This is a snapshot of the configuration of my machine running on AWS.

Copy Module

The copy module is often used in writing playbooks when we want to copy a file from a remote server to destination nodes.

For example, suppose we want to copy a file from a remote server to all destination machines.

ansible test-servers -m copy -a 'src=/home/knoldus/Personal/blogs/blog3.txt dest=/tmp' -u ec2-user

Yum Module

We use the Yum module to install a service.

ansible test-servers -m yum -a 'name=httpd state=present' -become -u ec2-user

Apache2 will be installed on our machines.

The key point to note here is that we have to use -become, which is new in version 2.6; before, we had to use -s.

Shell Module\*

When we want to run UNIX commands then we use shell module

ansible test-servers -m shell -a 'ls -la' -u ec2-user

https://gist.github.com/slathia15/be3f84fa101ab39fb0d1969b8a99fe5d

This will display all the files present in our machine with their permissions.

Service Module

When we want to ensure the state of a service that is service is running we use the service module.

ansible test-servers -m service -a 'name=httpd state=started' -become -u ec2-user

https://gist.github.com/slathia15/339cc8f6784bdec5037481f7dc225bbb

Apache2 is up on my machine.

Debug Module

To print a msg on hosts we use Debug module.

ansible test-servers -m debug -a 'msg=Hello' -u ec2-user

https://gist.github.com/slathia15/d408ac54c5cc1cddbf07d6b14abcaa3b

Hello, a message is printed on my machine.

Template Module

The Template module is used to copy a configuration file from the local system to the host server. It is the same as the copy module, but it dynamically binds group variables defined by us.

Here, I have vars in my source machine.

Include Module

When we want to include another playbook in our playbook, then we use the Include module.

User Module

To add a particular user to our module we can use User module. Here, we have added a user named Sachin to our module.

* **static and dynamic inventory model Ansible?**

**static** :In Ansible, a static inventory file is a plain text file that contains a list of managed hosts declared under a host group using either hostnames or IP addresses.

**Dynamic:**

In a configuration – especially a cloud setup such as AWS where the inventory file keeps constantly changing as you add or decommission servers, keeping tabs on the hosts defined in the inventory file becomes a real challenge. It becomes inconvenient going back to the host file and updating the list of hosts with their IP addresses.

* **PUBLIC SUBNET** If a **subnet's** traffic is routed to an internet gateway, the **subnet** is known as a **public subnet**. **PRIVATE** SUBNETIf a **subnet** doesn't have a route to the internet gateway, the **subnet** is known as a **private subnet**.
* **Terraform refersh command**
* The terraform refresh command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure. This can be used to detect any drift from the last-known state, and to update the state file. This does not modify infrastructure, but does modify the state file.
* **What are the basic component of deployment file?**
* In the .yaml file for the Kubernetes object you want to create, you'll need to set values for the following fields:

apiVersion - Which version of the Kubernetes API you're using to create this object

kind - What kind of object you want to create

metadata - Data that helps uniquely identify the object, including a name string, UID, and optional namespace

spec - What state you desire for the object

* **Git squash?**
* Git squash is a technique that helps you to take a series of commits and condense it to a few commits.

For example, assume that you have a series of n commits. By squashing you can make all the n-commits to a single commit.

We can do so by starting an Interactive Rebase session:

$ git rebase -i HEAD~3

An editor window will then open where you can choose how you want to manipulate the selected part of your commit history. Keep in mind that Interactive Rebase allows to perform many different actions on your commit history; for our example case here, however, we are interested in the "squash" action keyword. If you mark one or more lines as "squash", they will be combined with the one above:

* **multistage docker file:**
* Building efficient docker images are very important for faster downloads and lesser surface attacks.

Builder pattern helps to build efficient images to some extent but involves complexity

Docker introduced multi-stage builds from version 17.05 to solve this problem

With multi-stage builds, we can have multiple stages and copies only layers which are necessary for the final image form the previous stage.

* **how to make lightweight docker file?**
* by using multistage dockerfile
* what is diff bet dry run and executing a playbook?

When you do dry run it only shows you whats going to change, the changes are not made yet.

* Where is the log data of containers is stored?

the **logs are stored** on the **Docker** host in JSON files by default. Var/lib

* **what are modules in Terraform?**

A Terraform module is a set of Terraform configuration files in a single directory. Even a simple configuration consisting of a single directory with one or more .tf files is a module. When you run Terraform commands directly from such a directory, it is considered the root module.

* **how do containers within a pod communicate**?
* **Containers** in a **Pod** share the same IPC namespace, which means they can also **communicate** with each other using standard inter-process communications such as SystemV semaphores or POSIX shared memory. **Containers** use the strategy of the localhost hostname for **communication within a pod**
* **How do pods communicate with each other?**
* Containers within a pod share an IP address and port space, and can find each other via localhost. They can also communicate with each other using standard inter-process communications. ... Containers in different pods have distinct IP addresses and cannot communicate by IPC
* **how do you restrict communicate between the pods Kubernetes?**
* You can limit communication to Pods using the Network Policy API of Kubernetes. The Kubernetes Network Policy functionality is implemented by different network providers, like Calico, Cilium, Kube-router, etc. Most of these providers have some added functionality that extends the main Kubernetes Network Policy API.
* **where is the logs of container stored in Kubernetes?**
* These **logs** are usually **stored** in files under the /var/**log** directory of the server on which the service runs. For most services, that server is the **Kubernetes** master node
* **difference between replica set and replication controller?**
* The **replica set** are also known as next generation **replication controller**. The only **difference between replica set and replication controller is** the selector types. The **replication controller** supports equality based selectors whereas the **replica set** supports equality based as well as **set** based selectors.
* To delete a specific resource, run the following command:

Terraform destroy -target=*resource\_type*.*resource\_name*

* For example, to delete just the VM in the configuration that you applied earlier, run this command:

Terraform destroy -target=opc\_compute\_instance.default

* **28. What are the different types of services in Kubernetes?**
* There are particularly 3 main services in k8s.
* They are: **1) Cluster IP 2) Node IP 3) External** (Literally, we call it as load balancer)
* Cluster IP is the default load balancer in k8s. The drawback is that any node within the cluster can connect to LB but from outside no one can connect. In Node port type, LB has access to outside world or even internet. If you had created PODs in your k8s, and want to have the load balancer with them then we need to connect to ELB of 3rd party like AWS.
* what is the basic diff between hosting your app on vm and on containers?
* Virtual machines and **containers** differ in several ways, but the primary **difference** is that **containers** provide a way to virtualize an OS so that multiple workloads can run on a single OS instance. With **VMs**, the hardware is being virtualized to run multiple OS instances.
* what is **namespace** in Kubernetes?
* Namespaces are Kubernetes objects which partition a single Kubernetes cluster into multiple virtual clusters. Each Kubernetes namespace provides the scope for Kubernetes Names it contains; which means that using the combination of an object name and a Namespace, each object gets an unique identity across the cluster.
* **What is label and selector in Kubernetes?**
* Via a label selector, the client/user can identify a set of objects. The label selector is the core grouping primitive in Kubernetes. The API currently supports two types of selectors: equality-based and set-based. A label selector can be made of multiple requirements which are comma-separated.
* **difference between cmd and run in dockerfile?**
* RUN and CMD are both Dockerfile instructions. RUN lets you execute commands inside of your Docker image. These commands get executed once at build time and get written into your Docker image as a new layer. ... CMD lets you define a default command to run when your container starts.

Use RUN instructions to build your image by adding layers on top of initial image.

Prefer ENTRYPOINT to CMD when building executable Docker image and you need a command always to be executed. Additionally use CMD if you need to provide extra default arguments that could be overwritten from command line when docker container runs.

* **diff between daemonset and StatefulSet in kubernetes**
* Statefulsets is used for Stateful applications, each replica of the pod will have its own state, and will be using its own Volume. DaemonSet is a controller that ensures that the pod runs on all the nodes of the cluster. If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.
* **Diff between WEBHOOKS and POLLSCM**
* The concept of polling is very simple: send a request for new events (specifically, Create, Retrieve, and Delete events, which signal changes in data) at a predetermined frequency and wait for the endpoint to respond. If the endpoint doesn’t respond, there are no new events to share.
* Similar to polling, webhooks provide your application a way of consuming new event data from an endpoint. However, instead of sending repeated requests for new events, you provide the endpoint with a URL, usually within the endpoint UI, which your application monitors. Whenever a new event occurs within the endpoint app, it posts the event data to your specified URL, updating your application in real-time.
* BE NICE… TO YOUR SERVERS
* While polling and webhooks both accomplish the same task, webhooks are far more efficient. Zapier found that over 98.5% of polls are wasted. In contrast, webhooks only transfer data when there is new data to send, making them 100% efficient. That means that polling creates, on average, 66x more server load than webhooks
* **Diff stratergy in Kubernetes:**
* **Recreate**: This will terminate all existing instances and new instances will be recreated with the latest changes.

|  |
| --- |
| spec: |
|  | replicas: 3 |
|  | strategy: |
|  | type: Recreate |

* **Rolling updates** allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources. ... In Kubernetes, updates are versioned and any Deployment update can be reverted to a previous (stable) version.
* **What is Ingress Controller?**
* Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster. However, in real-world Kubernetes deployments, there are frequently additional considerations beyond routing for managing ingress.

For example:- For particular URIs, the new version of APIs is called for some older version is called. With the URI based routing, we have forwarded the requests to different versions of the application.

* **When would you use a ConfigMap?**
* Use a ConfigMap to keep your application code separate from your configuration. It is an important part of creating a Twelve-Factor Application. This lets you change easily configuration depending on the environment (development, production, testing) and to dynamically change configuration at runtime.
* **ConfigMaps**
* A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

* **COPY and ADD** are both **Dockerfile** instructions that serve similar purposes. They let you **copy** files from a specific location into a **Docker** image. **COPY** takes **in a** src and destination. ... A valid use case for **ADD is** when you want to extract a local tar file into a specific directory in your **Docker** image
* Top of Form
* **what is stateless and stateful application?**
* A **Stateless app** is an **application** program that does not save client data generated in one session for use in the next session with that client. A **Stateful application** saves data about each client session and uses that data the next time the client makes a request.
* A **ConfigMap** is an API object that lets you store configuration for other objects to **use**. Unlike most **Kubernetes** objects that have a spec , a **ConfigMap** has data and binaryData fields. These fields accept key-value pairs as their values. Both the data field and the binaryData are optional.
* What is init container?
* Init Containers are containers that run before the main container runs with your containerized application. They normally contain setup scripts that prepares an environment for you containerized application. Init Containers also ensure the wider server environment is ready for your application to start to run