23-Dec-2024

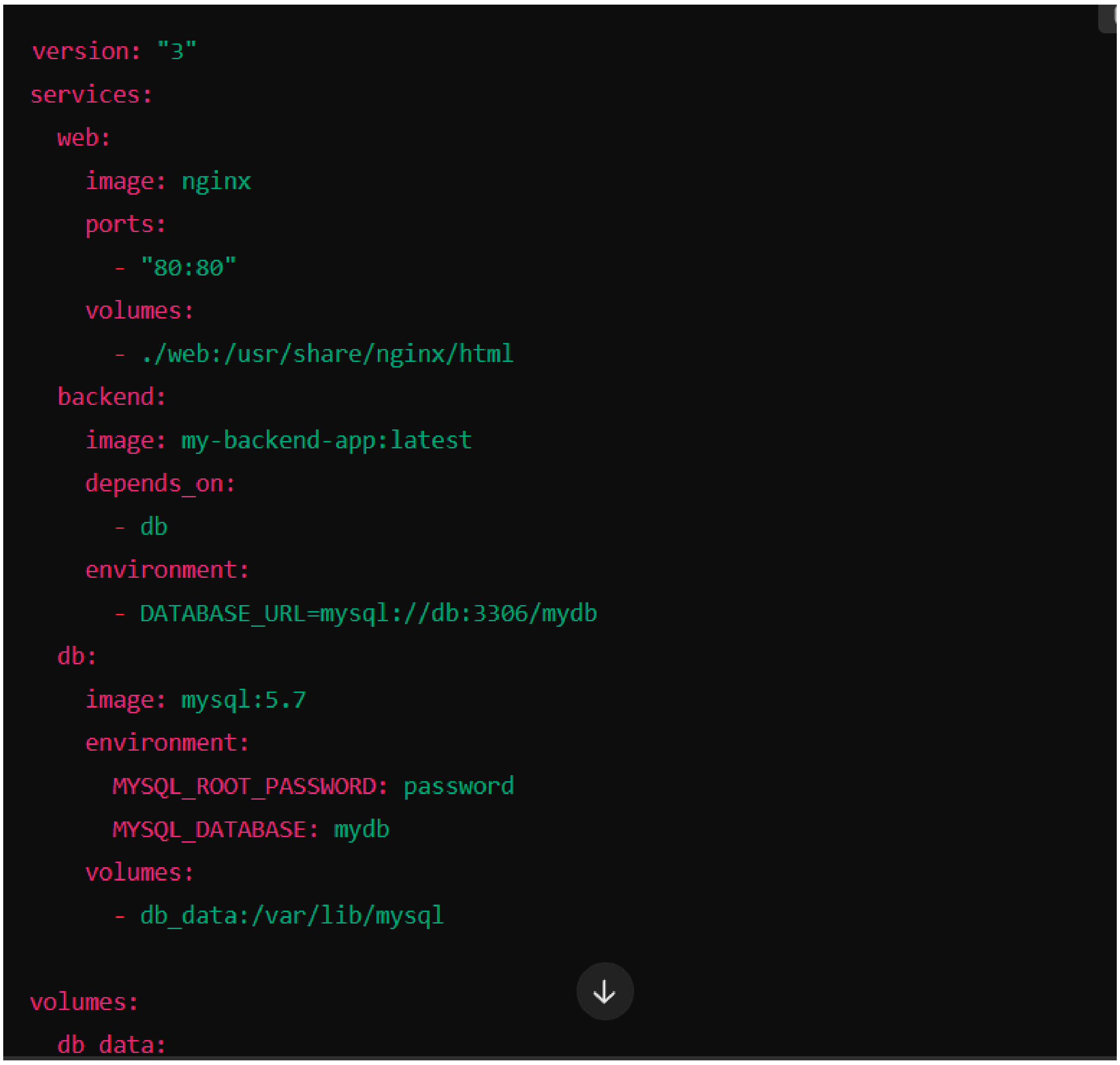
Internship Day - 106 Report:

How Docker Compose Works

To get a better understanding of how Docker Compose operates, let's go through a typical workflow:

1. Defining Services in docker-compose.yml

The first step is to define the services your application needs in a docker-compose.yml file. Here's an example for a web application that consists of a frontend, backend, and a database:



```yaml
version: "3"
services:
  web:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./web:/usr/share/nginx/html
  backend:
    image: my-backend-app:latest
    depends_on:
      - db
    environment:
      - DATABASE_URL=mysql://db:3306/mydb
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: mydb
    volumes:
      - db_data:/var/lib/mysql

volumes:
  db_data:
```

In this example:

- The web service uses an Nginx image and maps the local ./web directory to Nginx's document root.
- The backend service is a custom application image, which depends on the db service and connects to it using an environment variable.
- The db service is a MySQL container, with persistent data storage using volumes.

Running the Application

- Once the docker-compose.yml file is ready, you can start the entire application using the command:

```
docker-compose up
```

Docker Compose will:

- Create a network for the services to communicate.
- Start each service in its own container.
- Display logs from each container in the terminal.

  You can also use the -d option to run the services in the background (detached mode):

```
docker-compose up -d
```

Managing Services

Docker Compose offers a wide range of commands to manage your services:

- **docker-compose down**: Stops and removes all the containers and networks created by docker-compose up.
- **docker-compose logs**: Displays logs for all services or a specific service.
- **docker-compose restart [service_name]**: Restarts a specific service.
- **docker-compose ps**: Lists the running services and their status.

Advantages of Docker Compose:

1. Simplified Multi-Container Setup

Docker Compose reduces the complexity of managing multiple Docker containers. By defining everything in a single file, you avoid having to manually manage each container individually.

2. Easy to Use and Share

Since Docker Compose configurations are stored in simple YAML files, they can be easily shared across teams and environments. This enables consistent setups for development, testing, and production environments.

3. Version Control

The docker-compose.yml file can be versioned along with the application's code. This provides traceability and reproducibility of the application's infrastructure over time.

4. Rapid Application Deployment

Using Docker Compose, applications can be deployed and tested locally in minutes. This accelerates the development cycle since the entire application stack can be brought up quickly with just one command.

5. Portability Across Environments

With Docker Compose, your application can run consistently across different environments (local, CI/CD pipelines, staging, production) without worrying about environment-specific differences.

6. Efficient Resource Management

Compose enables you to define resource constraints for each service, such as CPU and memory limits. This ensures that no service consumes excessive resources, improving stability and performance.

Use Cases for Docker Compose

- **Local Development Environments**
  Compose is often used to set up local development environments for microservices-based applications. Developers can easily spin up all the services their application depends on with a single command, making the development process more efficient.

- **Continuous Integration (CI) Pipelines**
  Docker Compose is commonly used in CI pipelines to test applications in an isolated environment. It ensures that all services, such as databases and external dependencies, are available during testing, without having to provision separate infrastructure.

- **Staging Environments**
  Compose is also useful for setting up staging environments that closely mirror production systems. This ensures that applications can be thoroughly tested before being deployed to production.

24-Dec-2024

Internship Day - 107 Report:

1. Check Docker Compose Version

```
docker-compose --version
```

This command checks the installed version of Docker Compose on your system. If Docker Compose is installed, it will return the version number.

2. Install Docker Compose (If not already installed)

```
sudo apt install docker-compose
```

Installs Docker Compose using the apt package manager on a Linux-based system (e.g., Ubuntu). You need sudo for elevated privileges since installation requires administrative rights.

3. Create a Project Directory

```
mkdir composefirst
```

This command creates a new directory called composefirst, which will contain your Docker Compose project files.

4. Navigate to the Project Directory

```
cd composefirst
```

Changes the current working directory to the newly created composefirst directory so you can start working inside it.

5. Create a Python File (app.py)

```
touch app.py
```

This command creates an empty file called app.py where you'll write the code for your Python application.

6. Write Python Code in app.py

```python
# Example Python code (Flask app)
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

This is a simple Flask application that returns "Hello, World!" when accessed. Save this code in your app.py file.

7. Create a requirements.txt File

```
touch requirements.txt
```

This command creates an empty requirements.txt file. This file will list the Python dependencies required for your project.

Example content for requirements.txt:

```
Flask==2.0.0
```

8. Navigate Back to Parent Directory

```
cd ..
```

After setting up the files in the composefirst directory, this command takes you back one level in the directory structure. This might be useful if you're organizing your project and need to return to the parent folder.

9. Create a Dockerfile

```
touch Dockerfile
```

Creates a Dockerfile where you define the instructions to build a Docker image for your Python application.

Example content for a Dockerfile:

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Explanation of the Dockerfile:

- FROM python:3.9-slim: Uses the official Python 3.9 image as the base.
- WORKDIR /app: Sets the working directory inside the container.
- COPY requirements.txt .: Copies the requirements.txt file into the container.
- RUN pip install -r requirements.txt: Installs the Python dependencies from requirements.txt.
- COPY . .: Copies all files from your local directory to the container.
- CMD ["python", "app.py"]: Runs the Python application when the container starts.

10. Run Docker Compose

```
docker-compose up
```

This command brings up all the services defined in your docker-compose.yml file. If you haven't created this file yet, you need to do so.

Example content for a docker-compose.yml file:

```
version: "3"
services:
  web:
    build: .
    ports:
      - "5000:5000"
```

- version: "3": Specifies the version of Docker Compose file format.
- services: Defines the services (containers) in your application.

- web: Defines a service called "web" that builds from the local Dockerfile and maps port 5000 on the host to port 5000 in the container.

11. Check Running Containers

```
docker ps
```

This command lists all the running Docker containers, showing details like container ID, name, image, and port mappings.

Additional Docker Commands:

Stop Docker Compose:

```
docker-compose down
```

Stops and removes all containers, networks, and volumes defined by Docker Compose.

Build Docker Image Without Running:

```
docker-compose build
```

This command builds the Docker image without starting the containers.

26-Dec-2024

Internship Day - 108 Report:

Introduction to Kubernetes

Kubernetes, often abbreviated as K8s, is an open-source platform designed for automating the deployment, scaling, and operation of application containers. Initially developed by Google, it is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes has become the de facto standard for container orchestration, providing powerful tools to manage containers efficiently in production environments.

At its core, Kubernetes allows developers to manage large-scale, complex applications by abstracting the underlying infrastructure and automating many of the tasks that would otherwise need manual intervention. It has become popular for managing microservices architectures, which consist of independently deployable services working together within the same application.

Why Use Kubernetes?

The rise of Kubernetes is linked to the growing adoption of containerization technologies like Docker. Containers provide a way to package applications and their dependencies together into a single, lightweight unit that can run consistently across different environments. However, while containers make it easier to deploy applications, managing them at scale can be difficult. That’s where Kubernetes comes into play.

Here are a few reasons why organizations use Kubernetes:

- **Container Orchestration**: Managing hundreds or thousands of containers manually is complex. Kubernetes automates tasks like scheduling containers across clusters, handling failures, and ensuring that applications run smoothly.
- **Scalability**: Kubernetes makes it easy to scale applications up and down automatically based on demand, ensuring efficient resource utilization.
- **Portability**: Kubernetes abstracts the underlying infrastructure, making it easier to move workloads between different environments (e.g., on-premises data centers, cloud platforms) without modifying application code.
- **Self-Healing**: Kubernetes has built-in mechanisms to recover from failures. It can automatically restart containers that fail, replace containers, and reschedule workloads across nodes to maintain high availability.
- **Declarative Configuration**: With Kubernetes, users declare the desired state of their application, and Kubernetes ensures that the current state matches it. This declarative model makes it easier to manage and automate complex deployments.

How Kubernetes Works

Kubernetes is built around a set of core components and concepts that help it manage containerized applications:

- **Clusters**: A Kubernetes cluster consists of a set of machines (nodes) that run containerized applications. A cluster typically has a **master node** that controls the state of the cluster, and **worker nodes** that run the applications.
- **Pods**: The smallest unit of deployment in Kubernetes is the **pod**. A pod typically contains one or more containers that share resources like networking and storage. Pods are ephemeral and can be replaced when they fail or are updated.
- **Services**: Kubernetes provides services to expose a set of pods to the network. This abstraction allows communication between different parts of the application and enables load balancing across pods.
- **Controllers**: Kubernetes uses controllers to manage the state of applications. A controller, such as a **Deployment**, ensures that the desired number of replicas (copies) of a pod are running at any time.
- **Namespaces**: Namespaces allow for multiple virtual clusters within the same physical cluster, helping with organization and resource management.
- **ConfigMaps and Secrets**: These are mechanisms for decoupling configuration information from the containerized applications, allowing developers to manage configurations and sensitive information more securely.

Advantages of Kubernetes

Kubernetes offers several advantages that make it a powerful tool for managing containerized applications:

1. Scalability

One of the major advantages of Kubernetes is its ability to scale applications based on demand. It uses horizontal pod autoscaling, which automatically adds or removes pods as traffic or resource usage increases or decreases. This elasticity allows for efficient resource utilization, saving infrastructure costs while maintaining performance during peak loads.

2. Portability and Flexibility

Kubernetes is platform-agnostic, meaning it can run on public clouds (like AWS, Google Cloud, Azure), private data centers, or hybrid environments. Its containerized architecture allows you to move workloads easily between different environments without changing the underlying application code. This flexibility makes Kubernetes ideal for modern, cloud-native applications where multi-cloud or hybrid-cloud strategies are adopted.

3. Self-Healing

Kubernetes automatically handles failures at both the container and node levels. If a container fails, Kubernetes restarts it. If a node fails, it reschedules the affected containers to healthy nodes. This self-healing capability reduces downtime and ensures higher availability of applications. Furthermore, Kubernetes monitors the health of the system and automatically kills or restarts pods that are not functioning properly.

4. Load Balancing and Service Discovery

Kubernetes handles service discovery and load balancing automatically. It assigns a DNS name to each set of pods, allowing other services within the cluster to discover and communicate with them without manual intervention. Additionally, Kubernetes distributes traffic evenly across containers using load balancing techniques, ensuring optimal performance and availability.

5. Efficient Resource Management

Kubernetes optimizes resource utilization through its scheduling algorithms, which ensure that containers are deployed on nodes that have the right amount of resources (e.g., CPU, memory) available. This prevents over-provisioning and underutilization of resources, helping organizations reduce infrastructure costs while improving performance.

6. Declarative Configuration and Automation

Kubernetes uses a declarative approach to configuration, which simplifies the management of complex applications. Instead of managing every detail manually, users define the desired state of the application, and Kubernetes takes care of the rest. This approach enables automation of common tasks such as scaling, rollbacks, and upgrades, reducing the operational burden on development and operations teams.

Use Cases of Kubernetes

- **Microservices Architecture**: Kubernetes excels at managing microservices-based applications, where each component of the application runs in its container and can scale independently. Kubernetes provides the tools to manage such applications at scale, making it ideal for companies adopting microservices.
- **DevOps and CI/CD**: Kubernetes integrates well with continuous integration and continuous delivery (CI/CD) pipelines. It automates the process of deploying and scaling applications in different environments (e.g., development, staging, production), making it easier for teams to deliver features and updates quickly.
- **Hybrid and Multi-Cloud Environments**: With the growing adoption of hybrid and multi-cloud strategies, Kubernetes provides a consistent platform for managing applications across different infrastructures. Its abstraction of the underlying infrastructure allows developers to deploy applications seamlessly in any environment.
- **Big Data and AI Workloads**: Kubernetes is also increasingly used in big data and AI/ML (machine learning) applications. It provides the infrastructure needed to orchestrate large-scale data processing and AI workloads, making it a preferred choice for data-driven enterprises.

Challenges with Kubernetes

While Kubernetes offers many advantages, it also comes with some challenges:

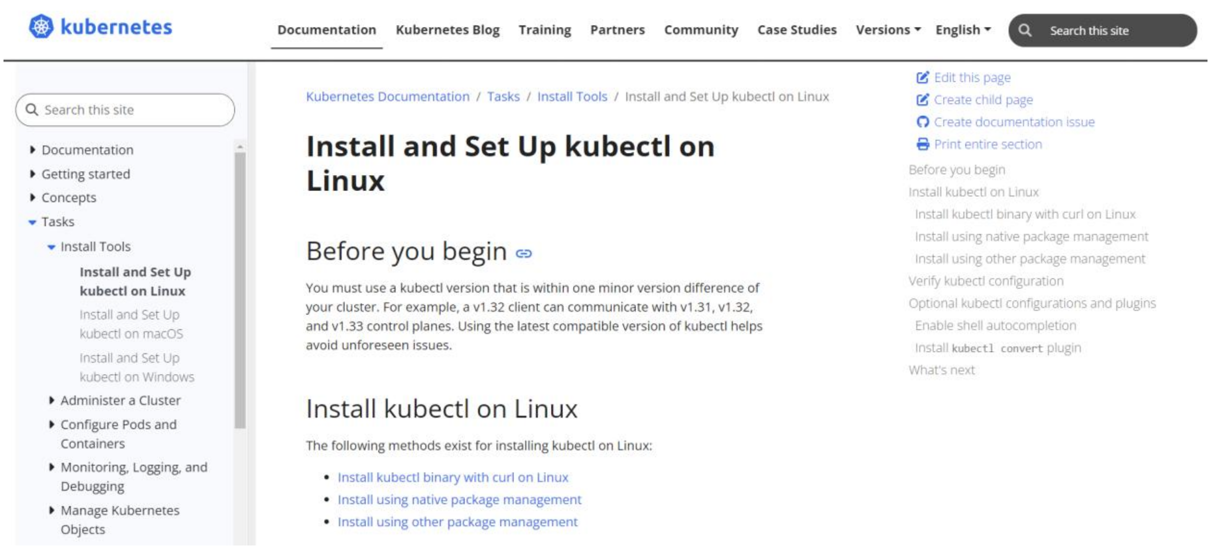
- **Complexity**: Kubernetes is a powerful but complex system that requires expertise to manage. Learning how to deploy and operate Kubernetes effectively can be a steep learning curve for teams without prior experience.
- **Networking Overhead**: Kubernetes introduces additional networking layers to manage communication between containers, which can result in overhead. Ensuring that network performance remains optimal in large-scale deployments can be challenging.
- **Security Management**: While Kubernetes provides tools to manage security (e.g., Role-Based Access Control, Secrets management), configuring it securely requires a deep understanding of both Kubernetes and the underlying infrastructure. Organizations must adopt best practices to avoid vulnerabilities.

27-Dec-2024

Internship Day - 109 Report:

INSTALLATION OF KUBERNETES

Open kubernetes interface



kubernetes — Documentation | Kubernetes Blog | Training | Partners | Community | Case Studies | Versions | English | Search this site

Search this site

- Documentation
- Getting started
- Concepts
- Tasks
  - Install Tools
    - Install and Set Up kubectl on Linux
    - Install and Set Up kubectl on macOS
    - Install and Set Up kubectl on Windows
  - Administer a Cluster
  - Configure Pods and Containers
  - Monitoring, Logging, and Debugging
  - Manage Kubernetes Objects

Kubernetes Documentation / Tasks / Install Tools / Install and Set Up kubectl on Linux

Install and Set Up kubectl on Linux

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.32 client can communicate with v1.31, v1.32, and v1.33 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

Install kubectl on Linux

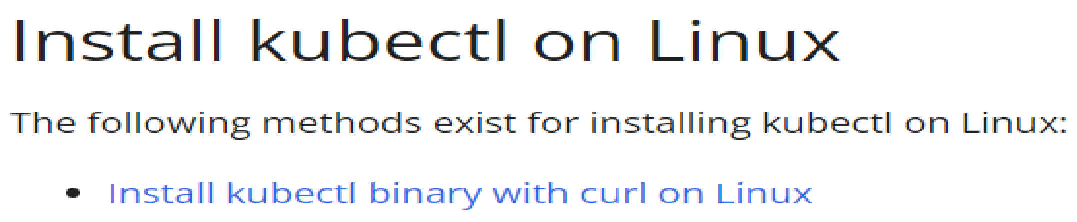
The following methods exist for installing kubectl on Linux:

- Install kubectl binary with curl on Linux
- Install using native package management
- Install using other package management

Edit this page
Create child page
Create documentation issue
Print entire section

Before you begin
Install kubectl on Linux
Install kubectl binary with curl on Linux
Install using native package management
Install using other package management
Verify kubectl configuration
Optional kubectl configurations and plugins
Enable shell autocompletion
Install `kubectl convert` plugin
What's next

Click on install kubectl binary with curl on linux

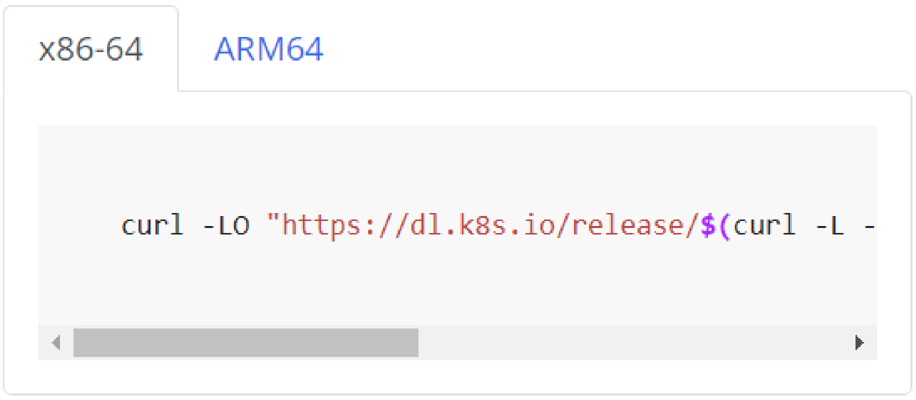


Install kubectl on Linux

The following methods exist for installing kubectl on Linux:

- Install kubectl binary with curl on Linux

Copy yhis command on gitbash & kubernetes installation get started


x86-64 ARM64

```
curl -LO "https://dl.k8s.io/release/$(curl -L -
```

Install **kops** (Kubernetes Operations)

- Kubernetes Operations (kops) is a tool to create, manage, and update production-grade Kubernetes clusters on cloud platforms.
- The user is asked to:
  - Open a command editor or terminal.
  - Search for how to install kops using a search engine (Google).

To install kops on Linux, you would typically follow these steps:

- **Download the latest version:**

```
curl -LO https://github.com/kubernetes/kops/releases/download/v1.XX.0/kops-linux-amd64
```

- **Make the binary executable:**

```
chmod +x kops-linux-amd64
```

- **Move it to a directory in your PATH:**

```
sudo mv kops-linux-amd64 /usr/local/bin/kops
```

5. Install Kubernetes Command-Line Tools

- This is likely referring to the kubectl installation. kubectl is a command-line tool that allows you to interact with your Kubernetes cluster.
- The notes suggest copying and pasting the necessary installation commands.

To install kubectl on Linux:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

6. Test the Installation of Kubernetes Tools

Command: kubectl version --client

- This command checks the version of the kubectl tool that was installed. It ensures that the client tool is working correctly.

Practical work through kubernetes

1. Install Secure Configuration Tool/Software

- **JDK** (Java Development Kit) installation: It seems the first step is about installing the necessary software, likely a secure connection tool or some dependency for running a secure shell (SSH) connection.
- **Step:** Download and install JDK or any other required software that is mentioned. Follow the installation steps based on your operating system.

2. SSH Setup

- Use the command: ssh -i <keyfile>.pem <username>@<IP address>
- **Keyfile:** This is the private key (.pem file) that allows SSH access to a server. You need to replace <keyfile> with your private key file and <IP address> with the actual public IP of the remote server.
- **Username:** Replace <username> with the user ID required to log in. Typically, for AWS instances, it might be something like ec2-user, ubuntu, or admin.
- **Example:**

```
ssh -i mykey.pem ec2-user@123.45.67.89
```

3. Create an IAM User (AWS Specific)

- Go to **AWS IAM** (Identity and Access Management) and create a new user with specific permissions. Make sure to:

- Assign the appropriate policies and roles for this IAM user, depending on the actions they need to perform.
- Make a note of the user’s credentials (Access Key ID, Secret Access Key).

4. Connecting via CLI (Command Line Interface)

- **AWS CLI setup:** You can use the AWS Command Line Interface to interact with AWS services directly from your terminal.

- To configure the CLI, you will need:

  - **Access Key ID**
  - **Secret Access Key**

This information is retrieved from the IAM user you created earlier.

- **Command to generate Access Key:** It seems to mention creating an access key by navigating to "Security Credentials" within the AWS Console.

5. Command-Line Interface (CLI) Setup

- Follow the prompts to input the access key and secret key, along with the default region and output format.
- **Example CLI Command for configuration:**

```
aws configure
```

During the process, you will be prompted to enter:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Default Region Name** (e.g., us-west-2)
- **Default Output Format** (e.g., json)

6. Download **.csv** File

- Download the security credentials (including access key and secret key) in a .csv file when creating the IAM user.
- Make sure to store it securely, as these credentials provide access to your AWS resources.

7. Next Steps (Optional Software)

- **Step 1: Install AWS CLI** using a command:

```
sudo apt install aws-cli --classic
```

- **Configure AWS CLI** as mentioned earlier.
- **Permissions:** Grant appropriate permissions to the user created earlier.

8. Search for **vpc** (Virtual Private Cloud)

- The next step is to search for information about VPCs, specifically Amazon VPC (Virtual Private Cloud), on AWS.
- AWS VPC allows you to provision logically isolated sections of the cloud where you can launch AWS resources in a virtual network.

9. Create a Hosted Zone on Route 53 (AWS)

- **Route 53** is Amazon's DNS (Domain Name System) web service.
- The step instructs to navigate to Route 53 and create a hosted zone. A hosted zone represents a domain and its DNS settings.
- This can be done via the AWS Management Console under the Route 53 service.

10. Kubernetes Cluster Creation

- The notes mention creating a Kubernetes cluster, specifying the "name" of the cluster as an important step.
- This would involve using kops or another tool to define and launch the cluster.

11. Configure Bastion Host

- **Bastion Host:** A Bastion host is a special-purpose server used to provide secure access to a private network from an external network, such as the internet.
- The configuration of a Bastion host could involve setting up a secure method of accessing resources in the private network, typically by SSH.

12. Sign In to **gitbash**

- Gitbash provides pre-packaged applications for deployment on cloud platforms or Kubernetes. The notes mention logging into gitbash, likely to access application packages or use gitbash tools.

13. Go to **My Accounts**

- Finally, the user is instructed to navigate to the "My Accounts" section, likely within gitbash or AWS, to manage their account details, settings, and possibly access keys or credentials.

14. Link Your Domain Name

- The first step involves linking your domain name. If you have a custom domain, you need to point it to the services you are using (for example, your Kubernetes cluster or an S3 website).
- This likely refers to setting up the DNS for the domain to ensure it resolves to the correct resources.

15. Go to DNS Record

- You'll need to access the DNS settings for your domain. This can usually be done through your domain registrar or a DNS management service like AWS Route 53.
- **DNS Records** are used to map domain names to IP addresses and other resources.

16. Add NS (Name Server) Record

- In the DNS settings, you are instructed to add an **NS (Name Server)** record.
- NS records specify which servers will handle DNS queries for your domain. You would add the NS record provided by your DNS host (like AWS Route 53) to ensure your domain resolves correctly.

17. Go Back to AWS

- Once the DNS configuration is complete, the next step is to return to the AWS Management Console.

18. Search for S3 (Simple Storage Service)

- **S3** is Amazon's storage service. You are instructed to search for S3 in the AWS Console.
- S3 is commonly used for storing files and hosting static websites.

19. Create an S3 Bucket

- The final step involves creating an S3 bucket.
- S3 buckets are storage containers in AWS where you can upload and store data. If you are hosting a static website or other resources, the bucket will store the files.

To create an S3 bucket:

- In the AWS Console, navigate to **S3**.
- Click **Create bucket**, name the bucket, and configure the settings (region, permissions, etc.).
- Once the bucket is created, you can upload your website files or other content.