Top DevOps Skills to learn in 2024

1) Docker and Kubernetes

Docker and Kubernetes form a powerful combination for implementing containerization and orchestration in a DevOps environment. Docker is used to create lightweight, portable containers, and Kubernetes is used to orchestrate and manage these containers, offering features like automatic scaling, rolling updates, and self-healing.

DevOps teams leverage Docker and Kubernetes to achieve several key objectives:

1. **Consistency:** Containers ensure that applications run consistently across different environments, addressing the "it works on my machine" problem.
2. **Scalability:** Kubernetes enables the automatic scaling of applications based on demand, ensuring optimal resource utilization.
3. **Portability:** Docker containers can run on any system that supports Docker, providing a high level of portability for applications.
4. **Efficiency:** Containerization reduces the overhead of virtualization, leading to more efficient resource utilization and faster application deployment.
5. **Resilience:** Kubernetes provides features like self-healing and automatic rollbacks, enhancing the resilience of applications in production environments.

2) GitOps

GitOps is a DevOps methodology that leverages version control systems, typically Git, to manage and automate the deployment and operation of infrastructure and applications. The core idea behind GitOps is to use Git as the single source of truth for both application code and infrastructure configurations. Here's how GitOps is used in DevOps:

1. **Infrastructure as Code (IaC):** GitOps encourages the use of Infrastructure as Code principles. Infrastructure configurations, including infrastructure definitions, policies, and configurations, are stored as code in a version-controlled repository. This ensures that changes to the infrastructure are tracked, auditable, and can be rolled back if needed.
2. **Application Delivery Automation:** Application code, deployment manifests, and configuration files are also stored in the version control system. Continuous Integration (CI) pipelines are set up to automatically build, test, and package applications. GitOps extends these CI/CD pipelines to also manage the deployment process using Git repositories.
3. **Declarative Infrastructure and Applications:** GitOps relies on a declarative approach, where the desired state of the infrastructure and applications is defined in configuration files. The Git repository serves as the source of truth for this desired state. Tools like ArgoCD or FluxCD continuously monitor the Git repository for changes and automatically apply those changes to the target environment to converge it to the desired state.
4. **Continuous Deployment:** Changes pushed to the Git repository trigger automated deployments. This ensures that the deployment process is reproducible and consistent across different environments. The deployment process is automated, reducing the risk of manual errors and providing a reliable and auditable process.

5. **Rollbacks and Auditing:** GitOps makes it easy to roll back changes in case of issues or errors. Since every change is versioned in Git, reverting to a previous known state is as simple as rolling back the Git commit. This also provides a clear audit trail, making it easier to track who made changes, when, and why.
6. **Collaboration:** GitOps promotes collaboration among development, operations, and other stakeholders by centralizing configuration and infrastructure code in a shared repository. Changes are proposed through Git pull requests, enabling peer reviews and discussions before changes are merged and applied.

3) Golang

Go is valued in DevOps for its simplicity, concurrency support, and performance, making it suitable for a wide range of tasks, from building command-line tools to developing complex distributed systems and microservices. Its strong ecosystem and growing community further contribute to its popularity in the DevOps space. Here are several ways in which Go is used in DevOps:

1. **Building Command-Line Tools:** Go is well-suited for building efficient and lightweight command-line tools. Many DevOps tools, such as Docker, Kubernetes, and Terraform, are written in Go. Go's statically linked binaries and cross-compilation support make it easy to distribute tools across different platforms without worrying about dependencies.
2. **Infrastructure as Code (IaC) Tools:** Go is often used to develop tools for managing infrastructure as code. For example, Terraform, a popular IaC tool for provisioning and managing infrastructure, is written in Go. Its

performance and concurrency features make it suitable for handling the parallel execution of infrastructure changes.

3. **Microservices and Service Mesh:** Go's support for concurrent programming and its low-latency performance make it a good choice for building microservices. Many organizations use Go to develop services that are part of their microservices architecture. Go is also utilized in projects like Istio, a service mesh for managing and securing microservices.
4. **Automation and Scripting:** DevOps processes often involve automation, and Go is used to write scripts and automation tools. Whether it's for deployment scripts, system monitoring, or custom automation tasks, Go's simplicity and efficiency make it a suitable choice.
5. **Containerization:** Go is widely used in the development of container-related tools. Docker, the popular containerization platform, has parts of its codebase written in Go. The language's ability to create statically linked binaries simplifies the distribution of container runtime components.
6. **Continuous Integration/Continuous Deployment (CI/CD) Tools:** Go can be used to build custom CI/CD pipelines or extend existing ones. Tools like Jenkins, which support the development of plugins in Go, demonstrate the language's applicability in the CI/CD domain.
7. **Networking Tools:** Go is commonly employed to build networking tools for tasks such as monitoring, packet analysis, and network automation. Its support for concurrent programming makes it well-suited for handling networking tasks efficiently.
8. **Monitoring and Logging Tools:** Go is used to develop monitoring agents, logging tools, and observability solutions. Its performance characteristics and ease of deployment make it a good fit for applications that require real-time monitoring and logging.

4) **Python**

Python is extensively used in DevOps for various tasks, ranging from automation and scripting to building infrastructure, managing configurations, and more. Here are some key areas in which Python is commonly employed in DevOps practices:

1. **Automation and Scripting:** Python's readability and ease of use make it a preferred choice for scripting and automation tasks. DevOps engineers use Python scripts to automate repetitive tasks such as configuration management, log parsing, file manipulation, and more.
2. **Configuration Management:** Tools like Ansible, SaltStack, and Chef leverage Python for writing configuration scripts and modules. These tools use Python to define the desired state of infrastructure and automate the configuration of servers and applications.
3. **Infrastructure as Code (IaC):** Python is used in IaC tools like Terraform and AWS CloudFormation. DevOps practitioners write scripts in Python to define and manage infrastructure resources in a declarative manner, enabling efficient provisioning and scaling.
4. **Continuous Integration/Continuous Deployment (CI/CD):** Python is often used in CI/CD pipelines for tasks such as test automation, building and packaging applications, and orchestrating deployment processes. Jenkins, a popular CI/CD tool, supports the use of Python scripts in pipeline definitions.
5. **Container Orchestration:** While Go is commonly associated with container orchestration tools like Kubernetes, Python is also used in this space. Open-source projects like Kubernetes provide Python client libraries and tools for managing and interacting with clusters.

6. **Monitoring and Logging:** Python is utilized for developing monitoring agents, log analysis tools, and other components of observability solutions. Libraries like Prometheus and tools like ELK Stack (Elasticsearch, Logstash, Kibana) can integrate with Python for monitoring and logging purposes.
7. **Web Development for Dashboards and Interfaces:** DevOps teams often build custom web interfaces and dashboards for monitoring and managing infrastructure. Python web frameworks like Flask and Django are used to create these interfaces, providing a way to visualize and interact with data.
8. **Cloud Automation:** Python is commonly employed in automating tasks within cloud environments. Cloud providers like AWS, Azure, and Google Cloud offer SDKs and APIs that can be accessed and managed using Python, allowing for infrastructure provisioning, management, and automation.
9. **Network Automation:** Python's simplicity and networking libraries make it a valuable tool for network automation tasks. Tools like Netmiko and NAPALM leverage Python to automate network device configuration and management.
10. **Security Automation:** Python is used for security-related automation tasks, including vulnerability scanning, compliance checks, and incident response. Security tools and frameworks often provide Python interfaces for integration.

5) **Terraform**

Terraform is a popular Infrastructure as Code (IaC) tool used in DevOps practices to automate the provisioning and management of infrastructure resources. Here are some key aspects of how Terraform is used in the DevOps workflow:

1. **Infrastructure Provisioning:** Terraform allows DevOps teams to define infrastructure as code using a declarative configuration language. This code describes the desired state of the infrastructure, specifying the resources needed (such as virtual machines, networks, and storage) and their configurations. Terraform then provisions and manages these resources across various cloud providers and on-premises environments.
2. **Multi-Cloud and Hybrid Cloud Support:** One of Terraform's strengths is its ability to work with multiple cloud providers and on-premises infrastructure seamlessly. DevOps teams can use a single set of Terraform configurations to manage resources across AWS, Azure, Google Cloud, and other supported platforms, facilitating multi-cloud and hybrid cloud deployments.
3. **Version Control Integration:** Terraform configurations are typically stored in version control systems such as Git. This enables versioning, collaborative development, and the ability to track changes over time. DevOps teams can manage infrastructure changes using the same version control workflows they use for application code.
4. **Automated Resource Lifecycle Management:** Terraform automates the lifecycle management of infrastructure resources. It can create, update, and delete resources based on changes in the configuration files. This ensures that the actual infrastructure aligns with the desired state defined in the Terraform code.
5. **Infrastructure Updates with Plan and Apply:** DevOps teams use Terraform's "plan" and "apply" commands to preview changes before they are executed. The "plan" command generates an execution plan, outlining the modifications Terraform will make to the infrastructure. The "apply"

command then applies these changes, updating the infrastructure accordingly.

6. **Dependency Management:** Terraform handles dependencies between resources automatically. It understands the relationships between different infrastructure components and ensures that resources are created or updated in the correct order. This simplifies the management of complex infrastructures with interdependent resources.
7. **Reusable Modules:** Terraform supports the creation of reusable modules, allowing DevOps teams to encapsulate and share infrastructure configurations. This promotes consistency and reusability across different projects and environments.
8. **Integration with Continuous Integration/Continuous Deployment (CI/CD):** Terraform is often integrated into CI/CD pipelines, enabling automated and repeatable infrastructure changes. DevOps teams can use Terraform to provision and manage infrastructure alongside application deployments, ensuring that both infrastructure and application changes are coordinated.
9. **State Management:** Terraform maintains a state file that records the current state of the infrastructure. This state file is crucial for tracking changes and ensuring that subsequent executions of Terraform commands are idempotent and can accurately determine the necessary modifications.

6) Grafana and Prometheus

These are the two widely used open-source tools in the DevOps and monitoring ecosystem. They are often used together to create powerful and comprehensive

monitoring and observability solutions. Here's an overview of each tool and how they work together:

1. **Prometheus**

Prometheus is designed for monitoring system and application metrics, providing real-time insights into the performance and health of the infrastructure. It is often used to collect metrics from various sources, such as applications, servers, and containers.

Type: Monitoring and alerting system.

Key Features:

1. Time-series database: Stores and queries time-series data.
2. Pull-based model: Collects metrics from instrumented targets at regular intervals.
3. Service discovery: Dynamically discovers and monitors services in the infrastructure.
4. Powerful query language (PromQL): Allows for flexible querying and analysis of metrics data.
5. Alerting: Supports the definition of alerting rules and notifications.

2. **Grafana**

Grafana complements Prometheus by providing a user-friendly interface for creating dashboards and visualizing metrics. It supports multiple data sources, and Prometheus is a popular choice for providing metric data to Grafana.

Type: Data visualization and dashboarding tool.

Key Features:

1. Dashboard creation: Allows users to create and customize dashboards with visualizations.
2. Data source integration: Connects to various data sources, including Prometheus.
3. Alerting: Supports alert rules and notifications based on data queries.
4. Plugins: Extensible with a wide range of plugins for additional data sources and features.

7) Ansible

Ansible is a powerful open-source automation tool used extensively in DevOps practices. It simplifies the process of configuring and managing servers, deploying applications, and orchestrating complex IT tasks. Here's how Ansible is commonly used in the DevOps workflow:

1. **Infrastructure as Code (IaC):** Ansible enables Infrastructure as Code by allowing you to define infrastructure configurations in YAML files (playbooks). This makes it easy to version control, share, and reproduce infrastructure setups.
2. **Automated Configuration Management:** DevOps teams use Ansible to automate the configuration of servers and networking devices. It ensures consistency across different environments and reduces the risk of configuration drift.

3. **Application Deployment:** Ansible facilitates the automated deployment of applications and services. Playbooks can be written to define the necessary steps for deploying, updating, and managing applications on servers.
4. **Task Automation:** Ansible is used for automating a wide range of IT tasks, from simple tasks like updating system packages to more complex tasks like user management, database configuration, and more.
5. **Idempotent Operations:** Ansible ensures idempotency, meaning that running a task multiple times results in the same state as running it once. This makes it safe to use in automation, as it doesn't make unnecessary changes.
6. **Agentless Architecture:** Ansible follows an agentless architecture, meaning that it doesn't require a client-side agent to be installed on managed systems. This simplifies the setup and makes Ansible easy to use.
7. **Inventory Management:** Ansible uses an inventory file to define the list of servers or network devices it will manage. This dynamic inventory can be easily extended to include new systems as the infrastructure evolves.
8. **Integration with Cloud Services:** Ansible provides modules to interact with various cloud services, allowing for the provisioning and management of cloud resources. It supports popular cloud providers such as AWS, Azure, Google Cloud, and others.
9. **Role-Based Execution:** Ansible allows you to organize tasks into roles, making it easy to reuse configurations across different projects. This modular structure enhances maintainability and scalability.
10. **Integration with Continuous Integration (CI) Tools:** Ansible can be integrated into CI/CD pipelines to automate the deployment process. This ensures that the infrastructure is provisioned and configured consistently with each code change.

11. **Community and Extensibility:** Ansible has a vibrant community and a wide range of pre-built roles and modules available on Ansible Galaxy. This makes it easy to leverage existing solutions and share best practices.