DEVOPS INSIDERS

Interview Guide

Tell me about Yourself?

My name is XXXXX, and I graduated from ABC college in 2014. I have a total of three years of professional experience. I started my career as a Linux Administrator, where I gained valuable knowledge and skills in managing Linux-based systems.

Currently, I am working as a DevOps engineer in my current company. As a DevOps engineer, my responsibilities revolve around maintaining infrastructure using tools like Terraform and managing resources in the Azure cloud environment. I also handle the configuration and maintenance of CI/CD pipelines, ensuring smooth and efficient deployment processes.

In addition, I am involved in deploying applications on AKS Kubernetes clusters, where I leverage containerization and orchestration techniques to ensure scalability and high availability. I am well-versed in monitoring and troubleshooting techniques to identify and resolve issues that may arise in the infrastructure or applications.

Tell me about Project?

Currently, I'm working on a project where we have a requirement to move a large ecommerce application from an on-premises server to the Azure Cloud. Our client was European based company, named Qiagen and they sell the genomic plates to scientists for research. The customer wants us to break down the application into smaller parts called microservices and run them on an AKS Cluster in Azure. To accomplish this, We have development, testing and devops teams. I'm part of the DevOps team, and we have designed a deployment architecture and a CI/CD process using Azure Pipelines.

Coming to the infrastructure, Our architect and the team have decided to follow the best practice provided by azure that is landing zone. So, We segregate the environment in 3 parts, dev, qa and prod. and each environment was having it's own subscription which was under a management group. We have adopted Terraform to provision and manage various resources such as VNet, subnets, AKS, ACR, Key Vaults, Application Gateway, virtual machines, databases, storage accounts, and Azure Bastion. We have created Terraform modules for these resources and organized the code in a repository. Additionally, we set up a CI/CD pipeline specifically for Terraform, which includes the init, plan, and apply of the Terraform code. We also incorporated a tfsec scan to check for any security vulnerabilities in the Terraform code.

Let me explain the deployment architecture we created. The starting point of our application is an application gateway, which acts as a security measure by enabling a Web Application Firewall. We also set up a custom domain and SSL certificate for added security. The application gateway handles the SSL offloading. Behind the application gateway, there are 12 microservices running inside the AKS cluster. To deploy the microservices, we created a deployment object for each of them with two replicas. We also set up a Kubernetes service of type clusterIP and an ingress object of type application gateway to expose the microservices. For managing secrets, we utilized Azure Key Vault, which was connected to AKS.

Now, let's talk about our CI/CD pipeline for one of the microservice. The pipeline consists of multiple stages. First, we pull the code from the repository, then we perform static code analysis using SonarQube. Next, we build the Docker image. Once the image is ready, we run a vulnerability scan on it using the Anchore scanner and push the image to Azure Container Registry (ACR). After that, the new image is deployed to AKS from the pipeline.

Tell me about Terraform Implementation?

Terraform Structure/Workflow:

- **We have two repositories: one for the Terraform code and another for modules.**
- **In the main branch of the Terraform code repository, we have three folders based on the environment: prod, dev, and stage.**
- **Additionally, we have a backup configuration stored in blob storage, and we use the azurerm provider.**
- **We have a Terraform CI/CD pipeline set up that automatically triggers and runs a plan when a pull request is made.**
- **When a merge occurs in the main branch, the pipeline applies the changes.**

What troubleshooting you have done?

Terraform State Lock:

Sometimes, in Terraform, the state can get locked, preventing any user or pipeline from making changes. To resolve this, you can use the "terraform force unlock" command followed by the state ID to release the lock. Additionally, you can also go to the Azure portal, navigate to the blob storage where the state is stored, and manually release the lock.

AKS Kubernetes CrashLoopBack:

When facing a CrashLoopBackOff error in AKS Kubernetes, there are several steps you can take to troubleshoot and resolve the issue:

Check the logs of the deployment: Review the logs of the application or the pod experiencing the CrashLoopBackOff. These logs can provide valuable information about any errors or issues that the application is encountering.

Share logs with developers: Once you have identified the errors in the logs, share them with the application developers. They can analyze the logs and provide insights into the root cause of the problem.

Rollback changes in Kubernetes: If the issue cannot be resolved quickly, it is advisable to rollback the current changes in Kubernetes. This ensures that the application is reverted to a stable state before investigating and addressing the underlying issue.

Day to Day Activities

Ticket-based Tasks: My day starts by addressing tickets and tasks assigned to me. These tasks can vary depending on the specific needs of the project and can include activities like provisioning infrastructure resources on the Azure cloud using Terraform, configuring and maintaining CI/CD pipelines, deploying applications on Kubernetes clusters, creating Docker images, and troubleshooting issues.

Deployment and Infrastructure Management: I actively participate in the deployment of applications on Kubernetes, ensuring that the necessary configurations and resources are in place for successful deployments. This involves setting up and maintaining infrastructure as code, utilizing tools like Terraform to provision and manage resources in an automated and consistent manner.

CI/CD Pipeline Maintenance: I am responsible for creating and maintaining CI/CD pipelines that automate the build, test, and deployment processes. This includes configuring the necessary steps, integrations, and automated tests to ensure a smooth and reliable release process.

Troubleshooting: Troubleshooting is a significant part of my day-to-day activities. I investigate and resolve issues that arise during deployments or in the production environment. This involves analyzing logs, identifying root causes, and collaborating with developers and other team members to implement effective solutions.

Monitoring: I monitor the performance and availability of applications and infrastructure using various monitoring tools. This helps me proactively identify potential issues, optimize performance, and ensure the stability of the system.

Collaboration with Developers: I work closely with developers to understand their requirements, provide necessary infrastructure support, and align the deployment process with their development practices. This collaboration ensures smooth integration between development and operations teams.

Git Concept

There are two types of branching strategies commonly used: Mono and Poly.

- In the Mono branching strategy, we typically have three main branches: Main (Production), Dev (Development), and Staging.
- In the Poly branching strategy, we may have multiple branches for different features or bug fixes.

CICD Configuration:

The CI/CD pipeline is configured to align with the branching strategy. Each branch triggers specific stages or actions based on the desired workflow.

Git Commands:

- git log: Displays a log of all commits in the repository.
- git commit -hash-: Refers to a specific commit using its hash.
- git push: Pushes local commits to the remote repository.

Branch Operations:

- git revert: Reverts a specific commit, undoing its changes in the repository.

Local changes can be moved into the Git stash using git stash to temporarily save them.

- git unstash moves the saved changes back to the local repository.

The stash is often used when switching branches to avoid conflicts and pull new changes.

Handling Conflicts:

When encountering conflicts during a merge, we utilize cherry-picking to selectively apply specific commits to resolve conflicts.

Fetching and Pulling:

- git fetch retrieves changes from the remote repository without merging them into the local branch.
- git pull combines git fetch with git merge, fetching and merging changes from the remote repository into the current branch.

Tell about your CICD process?

CICD Pipelines:

We utilize Azure Pipelines with YAML files for our CICD (Continuous Integration and Continuous Deployment) pipelines. We have two types of pipelines: one for Terraform and the other for applications.

Terraform Pipelines:

For Terraform pipelines, any changes made in the main branch or pull requests trigger the pipeline to apply the changes. These pipelines handle infrastructure provisioning and management using Terraform.

Application Pipelines:

We have multiple application pipelines tailored to different components such as frontend, backend, and others. These pipelines are responsible for building Docker images and deploying the applications onto Kubernetes clusters.

Branches in Application Repository:

In the application repository, we maintain different branches such as main (production), dev, and stage. Whenever a developer creates a merge on the dev branch, the corresponding pipeline is triggered.

CI Process:

During the CI (Continuous Integration) process, the pipeline clones the repository, builds the application, performs testing, scans for vulnerabilities, creates a Docker image, and pushes the image to an Azure container registry.

CD Process:

In the CD (Continuous Deployment) process, the pipeline deploys the application onto an AKS (Azure Kubernetes Service) cluster, ensuring a smooth deployment process for the applications.

Terraform Pipeline Behavior:

The Terraform pipeline is designed to trigger only on the main branch, ensuring that infrastructure changes are applied and propagated consistently.

What are the different services of Azure you have worked on?

Virtual Network (VNet), Load Balancer (LB), and Auto Scaling: Setting up and configuring virtual networks, load balancers for distributing traffic, and implementing auto scaling to handle varying workloads.

Virtual Machines (VM) and Bastion Host: Creating and managing virtual machines in Azure and utilizing a bastion host for secure remote access to VMs.

Azure Storage: Working with different types of Azure storage services, including Blob storage for unstructured data, Queue storage for reliable messaging, File storage for file sharing, and Table storage for NoSQL data storage.

Web Apps and App Services: Deploying and managing web applications using Azure Web Apps and App Services, which provide scalable hosting options for various types of applications.

Azure Kubernetes Service (AKS): Orchestrating and managing containerized applications using AKS, which simplifies the deployment, scaling, and management of containerized workloads.

Azure AD (Active Directory): Integrating applications with Azure AD for identity and access management, enabling secure authentication and authorization mechanisms.

Azure SQL: Deploying and managing Azure SQL databases, a fully managed relational database service in Azure that provides high availability, scalability, and security for your applications.

Cosmos DB: Working with Azure Cosmos DB, a globally distributed, multi-model database service, to develop applications with low latency, high throughput, and elastic scalability.

Azure Monitor: Utilizing Azure Monitor to collect and analyze telemetry data, gain insights into application performance, and set up alerts and notifications for proactive monitoring.

Azure Policy, Firewall, Network Security Groups (NSG), and Resource Groups (RG): Implementing Azure Policy to enforce compliance and governance, configuring Azure Firewall for network security, defining NSGs for network traffic filtering, and organizing resources using RGs.

Application Gateway and Front Door: Configuring Application Gateway and Front Door to manage and optimize web traffic, enable SSL offloading, and provide application-level routing and load balancing.

AKS Setup Details

In our setup, we utilize AKS (Azure Kubernetes Service) as a managed cluster, leveraging its benefits as a fully managed Kubernetes service. Within AKS, we have two types of worker nodes configured to cater to the specific needs of our applications.

The first worker node type is designed with high memory configuration, featuring 32GB RAM and 8 CPUs. This configuration is ideal for frontend applications that require more memory resources.

The second worker node type is optimized for high CPU performance, offering 16 CPUs and 16GB RAM. This configuration is specifically tailored for backend applications that demand more processing power.

To manage the assignment of workloads to the appropriate worker nodes, we use selectors in the deployment configurations. This ensures that frontend applications are scheduled on nodes with higher memory resources, while backend applications are assigned to nodes with higher CPU capabilities.

For scalability and auto-scaling capabilities, we have enabled the Horizontal Pod Autoscaler (HPA) for all deployments. This feature automatically adjusts the number of pods based on resource utilization, ensuring efficient utilization of resources and optimal performance.

To expose our applications to the external world, we leverage Ingress, which is deployed using a Helm chart. Ingress allows us to define routing rules and manage external access to our services.

To maintain separation and isolation, we utilize different namespaces for each application. This helps in organising and managing resources more effectively within the cluster.

To enforce access control, we implement RBAC (Role-Based Access Control) rules. Developers are granted read-only access to specific resources, ensuring they have the necessary visibility without compromising security or making accidental changes.

The deployment process for applications on AKS is integrated into our CI/CD pipeline, enabling automated and streamlined deployments.

To monitor the health and performance of our applications, we rely on Azure Monitor. This comprehensive monitoring solution provides insights into various metrics and enables proactive identification and resolution of issues.

Additional Questions to prepare

1. **Can you explain the process you followed to move the ecommerce application from an on-premises server to the Azure Cloud?**

Answer: First, we assessed the application's architecture and identified the components that could be broken down into microservices. We then provisioned an AKS cluster on Azure and containerized the microservices using Docker. Next, we set up a deployment architecture with an application gateway for security and SSL offloading. Finally, we established a CI/CD pipeline using Azure Pipelines to automate the deployment process.

2. **How did you break down the application into microservices? What factors did you consider while deciding on the granularity of microservices?**

Answer: We considered factors such as functional boundaries, scalability, and ease of maintenance while breaking down the application into microservices. We identified independent and cohesive components that could be decoupled and scaled independently. We also aimed to minimize inter-service dependencies to ensure each microservice could be developed, deployed, and maintained separately.

3. **What is an AKS Cluster in Azure, and why did you choose it as the deployment platform for the microservices?**

Answer: AKS (Azure Kubernetes Service) is a managed container orchestration service in Azure. We chose AKS because it provides a scalable and reliable platform for deploying, managing, and scaling containerized applications. AKS simplifies the management of Kubernetes clusters, automates scaling, and offers integration with other Azure services like Azure Container Registry and Azure Key Vault.

4. **Can you describe the deployment architecture you designed for the application on Azure? How does it ensure security and scalability?**

Answer: Our deployment architecture consists of an application gateway acting as a security measure with a Web Application Firewall. It handles SSL offloading and provides a custom domain and SSL certificate for added security. Behind the application gateway, we have an AKS cluster hosting the microservices. This architecture ensures secure communication between the clients and the microservices and enables horizontal scalability by allowing us to scale the AKS cluster based on demand.

7. Why did you choose Terraform for provisioning and managing Azure resources? Can you explain how you structured your Terraform code repository?

Answer: We chose Terraform because it provides a declarative approach to infrastructure provisioning and management. It allows us to define infrastructure as code and version control the configuration. Our Terraform code repository is organized using modules, where each module represents a specific resource or set of resources. This modular approach enables reusability, easy maintenance, and separation of concerns.

8. How did you incorporate a CI/CD pipeline specifically for Terraform? What were the stages in the pipeline, and what did each stage accomplish?

Answer: We set up a CI/CD pipeline using Azure Pipelines for our Terraform infrastructure code. The stages in the pipeline included: (1) Init: Initializing the Terraform environment and installing dependencies. (2) Plan: Generating an execution plan to preview the changes to be applied. (3) Apply: Applying the changes to provision or update the Azure resources. Each stage ensured the Terraform code was properly validated, planned, and executed.

9. Can you explain the role of Azure Key Vault in managing secrets for AKS? How did you ensure secure access to the secrets from the microservices?

Answer: Azure Key Vault plays a crucial role in securely storing and managing secrets for AKS. We integrated Azure Key Vault with AKS to store sensitive information such as database credentials, API keys, and certificates. Access to the Key Vault is tightly controlled using Azure Active Directory (AAD) authentication and authorization. We configured appropriate permissions, including role-based access control (RBAC), to ensure only authorized microservices could access the secrets.

10. Walk us through the CI/CD pipeline for one of the microservices. Which tools and stages were involved, and what was the purpose of each stage?

Answer: Our CI/CD pipeline for a microservice involved several stages. Firstly, we pulled the code from the repository using Azure Pipelines. Then, we performed static code analysis using SonarQube to identify code quality issues. Next, we built a Docker image using the code and dependencies. We ran a vulnerability scan on the image using the Anchore scanner to ensure it met security standards. Finally, we pushed the image to Azure Container Registry (ACR) and deployed it to AKS. The purpose of each stage was to ensure code quality, security, and seamless deployment to the target environment.

11. How many nodes did you have in your AKS cluster, and what factors influenced your decision on the cluster size?

Answer: We had a total of X nodes in our AKS cluster. The decision on the cluster size was influenced by factors such as the expected workload, resource requirements of the microservices, desired performance, and scalability needs. We considered the number of replicas for each microservice, the anticipated traffic, and the resources needed to handle peak loads.

12. Can you explain the scaling strategy you implemented for the AKS cluster? How did you ensure the cluster could handle varying workloads?

Answer: We implemented horizontal scaling for the AKS cluster using Kubernetes' built-in scaling features. We used metrics such as CPU utilization and incoming request rates to determine when to scale up or down. We configured autoscaling based on these metrics, ensuring that the cluster could dynamically adjust the number of nodes to handle varying workloads efficiently.

13. How did you monitor the performance and health of the AKS cluster and the microservices running within it?

Answer: We utilized Azure Monitor and enabled Azure Container Insights for comprehensive monitoring of the AKS cluster and microservices. Azure Monitor provided us with a centralized platform to collect, analyze, and visualize metrics and logs. We configured Azure Monitor to capture cluster-level metrics, such as CPU and memory utilization, node health, and network traffic. We also enabled Azure Container Insights, which allowed us to gain insights into the performance and health of individual containers, monitor application logs, and set up alerts for critical events. With Azure Monitor, we had a holistic view of the AKS cluster's performance and the ability to proactively identify and troubleshoot any issues.

14. What is a landing zone in the context of Azure infrastructure? How did you utilize it for segregating the dev, qa, and prod environments?

Answer: In Azure, a landing zone is a best-practice approach for setting up a well-architected environment. We utilized the landing zone concept to segregate our environments. We created separate subscriptions for the dev, qa, and prod environments, each under a management group. This allowed us to isolate resources, apply different access controls and policies, and manage costs effectively for each environment.

What challenges did you face while migrating the ecommerce application to Azure and implementing the DevOps process?

Answer: During the migration and DevOps implementation, we encountered a few challenges. One significant technical challenge was optimizing and fine-tuning the performance of the AKS cluster. As the number of microservices and their traffic increased, we needed to ensure that the cluster could handle the load efficiently. We faced issues related to resource utilization, network bottlenecks, and scaling strategies. It required careful monitoring, analysis, and adjustments to the cluster configuration, such as adjusting resource requests and limits, configuring appropriate pod-to-pod networking, and optimizing scaling thresholds. By closely working with the DevOps and development teams, we were able to identify and address these challenges to achieve optimal performance and scalability in the AKS cluster.

Another technical challenge we faced was in implementing a multi-stage Dockerfile for building the microservice images. While multi-stage builds provide benefits like smaller image sizes and improved build times, configuring the stages correctly and ensuring seamless transitions between them required careful consideration. We had to manage dependencies, handle environment configurations, and optimize the build steps. Debugging and troubleshooting issues that arose during the multi-stage build process also posed a challenge. By experimenting, fine-tuning the build steps, and collaborating with the development team, we overcame these challenges and successfully implemented efficient multi-stage Dockerfile builds.

One significant technical challenge was related to the automatic apply of Terraform changes in the CI/CD pipeline. Initially, we had configured the pipeline to automatically apply Terraform changes without manual intervention. However, this led to unexpected issues and potential risks, especially when changes had far-reaching impacts on the infrastructure. We faced situations where unintentional changes in the Terraform code caused disruptions in the production environment. To mitigate this, we introduced a manual review and approval stage in the pipeline. This allowed the DevOps team to carefully review and validate the Terraform changes before applying them to the target environment. By incorporating this manual stage, we ensured better control and reduced the possibility of unintended consequences.

How to troubleshoot VM?

Answer:

1. Check port using telnet
2. Check NSG rules
3. Check credentials access
4. Check Logs in Monitoring OS/VM
5. Resource Health - CPU/DISK
6. Windows - Disable RDP - Solve using RUN
7. VNET Firewall Rules