**AWS DevOps Interview Preparation Chart (5 Years Experience)**

| **Category** | **Topics to Cover** | **Hands-on Practice & Resources** |
| --- | --- | --- |
| **1. AWS Core Services** | - EC2, S3, VPC, IAM, Route 53, RDS, CloudFront, ALB/NLB | - Launch EC2, configure VPC & security groups, setup Route 53 |
|  | - AWS Auto Scaling, Load Balancers | - Create an Auto Scaling Group & test scaling policies |
| **2. Infrastructure as Code (IaC)** | - CloudFormation vs Terraform | - Write a Terraform script for an EC2 & S3 bucket |
|  | - Ansible, AWS CDK, Packer | - Configure an Ansible playbook for EC2 setup |
| **3. CI/CD Pipelines** | - Jenkins (Freestyle, Pipeline, Multi-Branch) | - Set up a CI/CD pipeline using Jenkins + GitHub |
|  | - AWS CodePipeline, CodeDeploy, GitHub Actions | - Deploy an app using AWS CodePipeline |
|  | - Blue-Green & Canary Deployments | - Implement Blue-Green deployment in AWS |
| **4. Containers & Kubernetes** | - Docker, Docker Compose | - Create a Dockerfile & push an image to ECR |
|  | - Amazon ECS vs EKS | - Deploy a sample app to ECS & EKS |
|  | - Helm, Kubernetes Networking, Ingress | - Install Helm & deploy a Helm chart |
| **5. Monitoring & Logging** | - AWS CloudWatch, CloudTrail | - Create CloudWatch alarms for EC2 & Lambda |
|  | - Prometheus, Grafana, Loki | - Configure Prometheus + Grafana for EKS |
|  | - ELK (Elasticsearch, Logstash, Kibana) | - Set up ELK stack on an EC2 instance |
| **6. Security & Compliance** | - IAM Roles, Policies, SCPs, SSM | - Set up IAM least privilege access |
|  | - AWS KMS, Secrets Manager | - Encrypt an S3 bucket & manage secrets |
|  | - Security Best Practices (NACLs, WAF) | - Implement AWS WAF on an ALB |
| **7. Networking & High Availability** | - VPC, Subnets, Security Groups vs NACLs | - Design a VPC with public & private subnets |
|  | - VPN, Direct Connect, Peering | - Set up a Site-to-Site VPN connection |
| **8. AWS Lambda & Serverless** | - AWS Lambda, API Gateway, DynamoDB | - Create a serverless app using Lambda & API Gateway |
|  | - EventBridge, Step Functions | - Automate event-driven workflows with Step Functions |
| **9. Debugging & Troubleshooting** | - CI/CD failures, EKS pod errors, EC2 slow performance | - Debug a failed Jenkins pipeline & fix it |
|  | - S3 access issues, IAM permission errors | - Troubleshoot a "403 Forbidden" S3 bucket issue |
| **10. Scenario-Based Problem Solving** | - Disaster Recovery & Backup Strategies | - Implement AWS Backup for RDS & EC2 |
|  | - Cost Optimization Techniques | - Use AWS Cost Explorer to analyze costs |
| **11. Interview Readiness** | - Behavioral Questions | - Prepare STAR-method responses for experience-based questions |
|  | - Whiteboarding & Architecture Design | - Design a scalable web app on AWS |

**Recommended Study Plan (4 Weeks Plan)**

**Week 1: AWS Core & Infrastructure as Code**

🔹 Review EC2, S3, IAM, VPC, CloudFormation, Terraform  
🔹 Hands-on: Deploy an EC2 instance & configure networking  
🔹 Hands-on: Write a Terraform script to deploy an AWS resource

**Week 2: CI/CD, Containers & Kubernetes**

🔹 Review Jenkins, AWS CodePipeline, GitHub Actions  
🔹 Hands-on: Create a Jenkins pipeline with AWS deployment  
🔹 Hands-on: Deploy a containerized app to EKS with Helm

**Week 3: Monitoring, Security & Networking**

🔹 Review CloudWatch, Prometheus, AWS Security Best Practices  
🔹 Hands-on: Set up CloudWatch alarms & dashboards  
🔹 Hands-on: Secure S3 with IAM policies & test access

**Week 4: Advanced Topics & Mock Interviews**

🔹 Practice Debugging & Troubleshooting AWS services  
🔹 Solve real-world DevOps case studies  
🔹 Conduct mock interviews (technical + scenario-based)

**AWS DevOps Real-Time Interview Hands-On Scenario-Based Questions**

Here are some practical, scenario-based AWS DevOps interview questions that test hands-on expertise:

**1️⃣ CI/CD & Automation**

🔹 **Scenario:** Your development team is pushing code changes frequently. They want an automated CI/CD pipeline in AWS. How will you set it up?  
✅ **Expected Answer:** Explain how to use **AWS CodePipeline** with **CodeCommit, CodeBuild, and CodeDeploy**. Mention **Jenkins integration, GitHub Actions**, and how you manage rollback in case of a failed deployment.

🔹 **Hands-on Task:**

* Create a **Jenkins pipeline** that automatically pulls code from GitHub and deploys it to an **EC2 instance**.
* Implement a **Canary Deployment** using AWS CodeDeploy.

**2️⃣ Infrastructure as Code (IaC) – Terraform & CloudFormation**

🔹 **Scenario:** Your team wants to provision AWS infrastructure in a repeatable and scalable way. How do you achieve this?  
✅ **Expected Answer:** Describe using **Terraform or AWS CloudFormation** to create **EC2 instances, VPCs, IAM roles**, and other resources.

🔹 **Hands-on Task:**

* Write a **Terraform script** to create an **S3 bucket, EC2 instance, and RDS database**.
* Deploy an **AWS CloudFormation stack** for a multi-tier web application.

**3️⃣ AWS Networking & Security**

🔹 **Scenario:** You deployed an application on **EC2 instances behind an ALB**, but some users cannot access it. How do you troubleshoot?  
✅ **Expected Answer:** Check **Security Groups, NACLs, ALB listener rules**, and use **AWS VPC Flow Logs** and **CloudWatch logs** for debugging.

🔹 **Hands-on Task:**

* Configure an **ALB with EC2 instances** and ensure **TLS termination**.
* Use AWS **WAF** to block a specific country’s IPs.

**4️⃣ Kubernetes & AWS EKS**

🔹 **Scenario:** You need to migrate a monolithic application to **AWS EKS**. What steps will you take?  
✅ **Expected Answer:** Explain how to containerize using **Docker**, create **Kubernetes manifests**, deploy them to **EKS**, and manage scaling with **Horizontal Pod Autoscaler (HPA)**.

🔹 **Hands-on Task:**

* Deploy a **3-tier application** (frontend, backend, database) on **AWS EKS**.
* Implement **Pod auto-scaling** and monitor it using **Prometheus & Grafana**.

**5️⃣ Monitoring & Logging – CloudWatch, Prometheus, ELK**

🔹 **Scenario:** Your production application experiences slow performance, and you need to diagnose the issue. What AWS tools will you use?  
✅ **Expected Answer:** Explain using **CloudWatch metrics, X-Ray for tracing, ELK (Elasticsearch, Logstash, Kibana), and Prometheus/Grafana** for deeper insights.

🔹 **Hands-on Task:**

* Set up **CloudWatch alarms** for high **CPU utilization** in EC2.
* Deploy **Prometheus and Grafana** on AWS EKS and configure **metrics for app monitoring**.

**6️⃣ AWS Lambda & Serverless**

🔹 **Scenario:** Your team wants to process **S3 file uploads automatically** without using an EC2 instance. What AWS service will you use?  
✅ **Expected Answer:** Use **AWS Lambda** with an **S3 event trigger** to process the file and store results in **DynamoDB/S3**.

🔹 **Hands-on Task:**

* Write a **Python Lambda function** triggered by S3 file uploads.
* Connect an **API Gateway** to a Lambda function and deploy a serverless API.

**7️⃣ High Availability & Disaster Recovery**

🔹 **Scenario:** Your application runs on a single AWS region, but the business wants it to be available **even if the region fails**. What will you do?  
✅ **Expected Answer:** Use **multi-region deployment** with **Route 53 failover routing, AWS Global Accelerator, and RDS Multi-AZ replication**.

🔹 **Hands-on Task:**

* Implement **Route 53 DNS failover** between two AWS regions.
* Configure **AWS RDS Read Replicas** in a secondary region.

**8️⃣ AWS IAM & Security Best Practices**

🔹 **Scenario:** You suspect someone is accessing AWS resources using an unauthorized IAM user. How will you investigate?  
✅ **Expected Answer:** Check **AWS CloudTrail logs**, enable **MFA**, restrict access using **IAM policies**, and use **AWS Config for compliance checks**.

🔹 **Hands-on Task:**

* Create an **IAM role with least privilege access** for an application.
* Enable **AWS GuardDuty** to detect unusual security activity.

**9️⃣ Cost Optimization in AWS**

🔹 **Scenario:** Your AWS bill has increased significantly this month. How do you analyze and reduce costs?  
✅ **Expected Answer:** Use **AWS Cost Explorer, Reserved Instances, Spot Instances, S3 lifecycle policies, and Auto Scaling** to optimize costs.

🔹 **Hands-on Task:**

* Set up **AWS Budgets & Cost Alerts** for EC2 usage.
* Configure **S3 lifecycle policies** to move old data to Glacier.

**🔟 Real-Time Troubleshooting Tasks**

✅ **Debugging Scenarios:**

1. **A Kubernetes pod is stuck in “CrashLoopBackOff” – How do you troubleshoot?**
2. **Your Terraform deployment failed due to a missing IAM role – How do you fix it?**
3. **Jenkins CI/CD pipeline failed at the deployment stage – What steps do you take?**
4. **AWS Auto Scaling is not launching new instances – How do you investigate?**
5. **An S3 bucket policy is not allowing public access despite settings – What do you check?**

**Final Thoughts 💡**

These hands-on **scenario-based AWS DevOps interview questions** will test your ability to **think practically and troubleshoot real-world problems**. If you’d like, I can also provide sample answers or guide you through a **mock hands-on exercise**. Let me know how you'd like to proceed! 🚀😊

Great! Let's break this down step by step.

**Scenario 1: CI/CD Pipeline with AWS CodePipeline & CodeDeploy**

**Step 1: Set Up CodeCommit Repository**

The first step is to set up a **CodeCommit repository** where your team will push the code. This will serve as the source stage for the pipeline.

1. **Create a CodeCommit Repository:**
   * Log in to the **AWS Management Console**.
   * Navigate to **CodeCommit** and click on **Create repository**.
   * Give the repository a name, e.g., MyAppRepo.

**Step 2: Set Up AWS CodeBuild (Build Stage)**

Next, you'll create a **CodeBuild** project to compile and test the application.

1. **Create a Build Project in CodeBuild:**
   * Navigate to **CodeBuild** in the AWS Management Console.
   * Click on **Create build project**.
   * Select **AWS CodeCommit** as the source provider and choose the repository MyAppRepo.
   * Choose an **Environment Image** (e.g., aws/codebuild/standard:5.0).
   * Add a **buildspec.yml** file to your repository, which defines the build steps (e.g., installation, testing, packaging). This file should be in the root of your repository.

Example buildspec.yml:

version: 0.2

phases:

install:

commands:

- echo Installing dependencies...

build:

commands:

- echo Building the application...

- # Add commands to build your app (e.g., npm install, mvn package, etc.)

post\_build:

commands:

- echo Build complete.

artifacts:

files:

- '\*\*/\*'

discard-paths: yes

**Step 3: Set Up CodeDeploy (Deploy Stage)**

You will now configure **CodeDeploy** to deploy the application to your **EC2 instances**.

1. **Create a CodeDeploy Application:**
   * Go to the **CodeDeploy** section in the AWS Console and click on **Create application**.
   * Choose the **Compute platform** as **EC2/On-premises**.
   * Give your application a name, e.g., MyAppDeployment.
2. **Create a Deployment Group:**
   * After creating the application, click on **Create deployment group**.
   * Choose **EC2 instances** as the deployment target.
   * Ensure your EC2 instances have the **CodeDeploy agent** installed and are tagged with the correct **deployment group** (e.g., MyAppInstances).
   * Under **Deployment type**, choose **In-place deployment**.

**Step 4: Create the CodePipeline**

Now, let's create the **CodePipeline** that ties everything together.

1. **Create CodePipeline:**
   * Go to **AWS CodePipeline** and click **Create pipeline**.
   * Set the **pipeline name**, e.g., MyAppPipeline.
   * In the **Source** stage, select **AWS CodeCommit** as the source provider and choose the repository MyAppRepo.
   * In the **Build** stage, choose **AWS CodeBuild** and select the build project you created earlier.
   * In the **Deploy** stage, choose **AWS CodeDeploy** and select the **deployment application** and **deployment group** you created.
2. **Review & Create:**
   * Review all the stages in the pipeline.
   * Click **Create pipeline**.

Once the pipeline is created, every time code is pushed to the repository, the pipeline will trigger automatically:

1. The **CodeCommit repository** serves as the source.
2. The code is **built** using **CodeBuild**.
3. The **application** is deployed to EC2 using **CodeDeploy**.

**Hands-on Task**

Now let's simulate a hands-on task!

1. **Task 1:** Create a simple **CodeCommit** repository and push some sample code to it.
   * Create a buildspec.yml file.
   * Ensure it has a build command that simulates building an app (e.g., echo "Build completed").
   * Push the code to **CodeCommit**.
2. **Task 2:** Set up **CodeBuild** with the repository and run the build.
   * Ensure that the build completes successfully with output in the **CodeBuild** logs.
3. **Task 3:** Set up **CodeDeploy** for EC2 instances.
   * Install **CodeDeploy agent** on your EC2 instances (if not already done).
   * Create a **deployment group** and ensure EC2 instances are correctly tagged.
4. **Task 4:** Create **CodePipeline** to automate the entire CI/CD process.
   * Tie the **source, build, and deploy** stages together.
   * Trigger the pipeline manually or automatically upon new code commits.

**Next Steps:**

Would you like me to guide you through any of these tasks specifically, or do you have questions about the setup? Let me know how you'd like to proceed!