**Task details**

You want to design a continuous delivery architecture for **a scalable and secure 3-tier Node application**.

Application to use can be found at <https://git.toptal.com/henrique/node-3tier-app2>

Both web and API tiers should be exposed to the Internet and the DB tier should not be accessible from the Internet.

You should clone the repository and use it as the base for your system.

* You need to create resources for all the tiers.
* The architecture should be completely provisioned via some infrastructure as a code tool.
* The presented solution must handle server (instance) failures.
* Components must be updated without downtime in service.
* The deployment of new code should be completely automated (bonus points if you create tests and include them in the pipeline).
* The database and any mutable storage need to be backed up at least daily.
* All relevant logs for all tiers need to be easily accessible (having them on the hosts is not an option).
* You should fork the repository and use it as the base for your system.
* You should be able to deploy it on one larger Cloud provider: AWS/Google Cloud/Azure/DigitalOcean/RackSpace.
* The system should present relevant historical metrics to spot and debug bottlenecks.
* The system should implement CDN to allow content distribution based on client location.

As a solution, please commit to the Toptal git repo the following:

* An architectural diagram / PPT to explain your architecture during the interview.
* All the relevant configuration scripts (Terraform/Ansible/Cloud Formation/ARM Templates)
* All the relevant runtime handling scripts (start/stop/scale nodes).
* All the relevant backup scripts.
* You can use another git provider to leverage hooks, CI/CD, or other features not enabled in Toptal's git. Everything else, including the code for the CI/CD pipeline, must be pushed to Toptal's git.

To design a continuous delivery architecture for the provided 3-tier Node application, we'll leverage AWS services and infrastructure as code tools such as Terraform for provisioning. Here's an outline of the architecture and the steps to achieve the requirements:

**### Architecture Overview:**

1. \*\*Networking:\*\*

- Create a Virtual Private Cloud (VPC) to host the application.

- Use public and private subnets. Place the web and API tiers in public subnets and the DB tier in a private subnet.

- Utilize Network Access Control Lists (NACLs) and Security Groups to control inbound and outbound traffic.

2. \*\*Compute:\*\*

- Web and API tiers:

- Deploy EC2 instances or utilize AWS Fargate for containerized applications.

- Use auto-scaling groups to handle server failures and scale based on demand.

- Database tier:

- Utilize Amazon RDS for a managed database service.

- Ensure the DB tier is placed in a private subnet and not exposed to the internet.

3. \*\*Storage:\*\*

- Use Amazon S3 for storing application artifacts and backups.

- Implement Amazon EFS for shared file storage if required.

4. \*\*Deployment Pipeline:\*\*

- Utilize AWS CodePipeline for continuous integration and continuous deployment (CI/CD).

- Implement AWS CodeBuild for building application artifacts.

- Integrate with GitHub for source code management.

5. \*\*Monitoring and Logging:\*\*

- Use Amazon CloudWatch for monitoring application metrics and logs.

- Implement centralized logging using Amazon CloudWatch Logs.

- Set up alarms for monitoring resource utilization and application health.

6. \*\*Backup and Restore:\*\*

- Use AWS Backup for automating backups of the database and any mutable storage.

- Store backups in Amazon S3 for durability and accessibility.

7. \*\*Content Delivery Network (CDN):\*\*

- Utilize Amazon CloudFront for CDN to cache and distribute content globally based on client location.

### Steps to Implement:

1. \*\*Infrastructure Provisioning:\*\*

- Define infrastructure as code using Terraform.

- Create Terraform modules for VPC, subnets, security groups, EC2/Fargate, RDS, S3, CloudFront, etc.

- Store Terraform configuration files in the repository.

2. \*\*Deployment Automation:\*\*

- Set up AWS CodePipeline to automate the deployment process.

- Configure GitHub webhook to trigger pipeline execution on code changes.

- Implement build and test stages in CodeBuild.

- Deploy updated artifacts to EC2/Fargate and RDS.

3. \*\*Backup Automation:\*\*

- Write scripts to automate backups of the database and storage to Amazon S3.

- Schedule backups to run daily using AWS Lambda or AWS Backup.

4. \*\*Monitoring and Logging:\*\*

- Configure CloudWatch metrics and alarms for monitoring resource utilization and application health.

- Set up CloudWatch Logs for centralized logging of application logs.

- Create custom dashboards in CloudWatch for visualizing metrics.

5. \*\*CDN Implementation:\*\*

- Set up Amazon CloudFront distribution to serve static and dynamic content.

- Configure caching behavior and distribution settings based on client location.

6. \*\*Documentation and Presentation:\*\*

- Create an architectural diagram or PowerPoint presentation to explain the architecture during the interview.

- Document configuration scripts, runtime handling scripts, and backup scripts.

- Commit all relevant files and scripts to the Toptal git repository.

By following these steps, you can design a scalable and secure continuous delivery architecture for the provided 3-tier Node application on AWS while meeting all the specified requirements.

git clone https://git.toptal.com/henrique/node-3tier-app2.git

cd node-3tier-app2

git remote -v

git remote rename origin old-origin

git remote add origin https://git.toptal.com/screening-ops/Arjun-kumar-kakinada-6.git

git push -u origin --all

git push -u origin –tags