Particle41 / TheAttic

devops-challenge-senior

Clone

A DevOps challenge for new candidates who want to join Particle41

main | Files | Filter files

/

Name	Size	Last commit	Message
README.md	8.45 KB	2025-05-23	Add Warning about cloud provider c

README.md

Welcome to the Particle41 DevOps Team Challenge

This challenge is for candidates who want to join the Particle41 DevOps team.

It is designed to assess your level of familiarity with common modern development and operations tools and concepts.

Warning: Working through this challenge WILL INCUR COSTS on your cloud account. Particle41 does not cover any costs associated with the challenge, so please ensure you clean up your cloud environment properly after you verify that your code works.

Summary

We aim to hire software engineers who embrace the DevOps mindset, especially taking an infrastructure-as-code approach to software and infrastructure deployment.

This challenge is designed to evaluate your abilities in the following technologies and concepts:

- Software development (in general), by creating an extremely minimal web service.

- Terraform, including writing a module and using it to deploy infrastructure.
- Documentation, including a short blurb about the purpose/contents of your repo as well as simple deployment instructions.

This assessment consists of two parts, with an extra-credit section at the end. The first part asks you to create a small application and containerize it. The second part asks you to create a terraform module to deploy a VPC, the necessary ECS/EKS/Lambda (or Azure/GCP equivalent) infrastructure required, and deploy the application to it.

When you have finished your challenge and your repository is ready for review, please tell us at careers@particle41.com. Remember to include the URL to your repository.

Documentation is MANDATORY

It is mandatory to include documentation for your repository explaining how to use it.

Imagine that someone with less experience than you will need to clone your repository and deploy your container, or deploy your terraform infrastructure.

With that in mind, you must provide all the instructions they will need to do that successfully. These must include any prerequisites for deployment; mention of needed tools and links to their installation pages; how to configure credentials for the tool of your choice; and what commands to run for deploying your code.

We want to see your ability to properly document and communicate about your work with the team.

- Add a `README.md` to the root directory of your project, with instructions for the team to deploy the projects you created. Include any notes (purpose, etc.) that you might add to the README if this were a real project.
- Publish your code to a public Git repository in a platform of your choice (e.g. GitHub, GitLab, Bitbucket, etc.), so that it can be cloned by the team.

A word about generative AI

It is ok to use generative AI to complete this challenge, but we want to be sure that you know what you're doing.

addresses some extra credit. Don't waste our time (and yours!) submitting a solution that doesn't work.

Task 1 - Minimalist Application Development / Docker / Kubernetes

Tiny App Development: 'SimpleTimeService'

- Create a simple microservice (which we will call "SimpleTimeService") in any programming language of your choice: Go, NodeJS, Python, C#, Ruby, whatever you like.
- The application should be a web server that returns a pure JSON response with the following structure, when its `/` URL path is accessed:

```
{
  "timestamp": "<current date and time>",
  "ip": "<the IP address of the visitor>"
}
```

Dockerize SimpleTimeService

- Create a Dockerfile for this microservice.
- Your application MUST be configured to run as a non-root user in the container.

Build SimpleTimeService image

- Publish the image to a public container registry (for example, DockerHub) so we can pull it for testing.

Push your code to a public git repository

- Push your code to a public git repository in the platform of your choice (e.g. GitHub, GitLab, Bitbucket, etc.). MAKE SURE YOU DON'T PUSH ANY SECRETS LIKE API KEYS TO A PUBLIC REPO!
- We have a recommended repository structure here.

container must run and stay running.

Other criteria for evaluation will be:

- Documentation: you MUST add a `README` file with instructions to deploy your application.
- Code quality and style: your code must be easy for others to read, and properly documented when relevant.
- Container best practices: your container image should be as small as possible, without unnecessary bloat.
- Container best practices: your application MUST be running as a non-root user, as specified in the exercise.

Task 2 - Terraform and Cloud: create the infrastructure to host your container.

Using Terraform, create the following infrastructure in AWS (or equivalent):

- If server-based:
  - A VPC with 2 public and 2 private subnets.
  - An ECS/EKS or equivalent cluster deployed to that VPC.
  - A ECS/EKS task/service resource to run your container
  - The tasks and/nodes must be on the private subnets only.
  - A load balancer deployed in the public subnets to offer the service.
- If serverless:
  - A VPC with 2 public and 2 private subnets.
  - A Lambda or equivalent function running your container
  - Appropriate configuration to associate the function with the private subnets.
  - An API Gateway, CDN, or loadbalancer to trigger your function

If you prefer, you may use popular modules from the Terraform registry (for example the VPC and EKS modules).

Push your code to a public git repository

- Push your code to a public git repository in the platform of your choice (e.g. GitHub, GitLab, Bitbucket, etc.). MAKE SURE YOU DON'T PUSH ANY SECRETS LIKE API KEYS TO A PUBLIC REPO!
- We have a recommended repository structure here.

---

and

```
terraform apply
```

must be the only commands needed to create the infrastructure and deploy the container.

You MUST NOT commit any credentials to the git repository. Instead, provide instructions in the README about how to authenticate to AWS to deploy the infrastructure.

Other criteria for evaluation will be:

- Code quality and style: your code must be easy for others to read, and properly documented when relevant.
- Terraform best practices: Use variables in your infrastructure root module, and provide some good defaults in a `terraform.tfvars` file.

Notes, Suggestions, and the opportunity to 'show off'!

Suggested Repo Structure

```
.
├── app <-- app files/directories and Dockerfile go here
└── terraform <-- Terraform files/directories go here (i.e. we will run
```

Extra Credit!

THIS SECTION IS *COMPLETELY OPTIONAL*! THERE IS NO PENALTY FOR IGNORING THIS!

Are you an overachiever? Demonstrate your mastery of cloud-native IaC tooling by doing any of these:

- Code to initialize and use a remote Terraform backend (S3 and DynamoDB) for state and locking instead of a local `.tfstate` file.