Merapar Technical Challenge Solution

Author: Alan Son

1. Introduction

On a cloud platform of my choice, I was tasked with provisioning a service using **Infrastructure as Code (IaC)** that serves an HTML page. The "dynamic string" portion should be modifiable at runtime **without requiring a redeploy**. The following document explains my rationale.

---

2. Chosen Solution: AWS Lambda and API Gateway

Rationale:
The solution architecture uses AWS Lambda for the core logic and Amazon API Gateway to expose a secure HTTP endpoint. The input string is stored in **AWS Systems Manager (SSM) Parameter Store**, enabling secure, version-controlled, and scoped access to configuration values. This approach aligns with modern cloud-native patterns and ensures minimal infrastructure overhead.

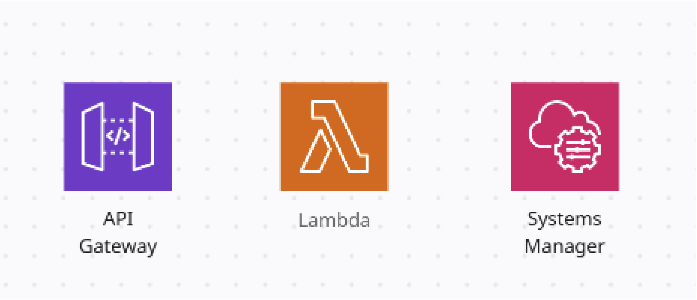
Pros:

- Serverless and scalable by default
- Minimal operational overhead
- Cost-effective with pay-per-request model
- Secure configuration using SSM Parameter Store
- Seamless integration with AWS IAM, CloudWatch, and GitHub Actions

Cons:

- Cold start latency may affect user experience for infrequent requests
- Resource limits on Lambda functions (memory, duration)
- Debugging locally is more involved than with containerized or VM-based applications

---



3. Alternative Solution Considered: EC2 with Docker and Nginx

Rationale:
This would involve deploying a Docker container on an EC2 instance with Nginx handling routing. It offers more environmental control but at the cost of increased maintenance, complexity, and reduced elasticity.

Pros:

- Full control over OS, container runtime, and HTTP stack
- Easy to replicate locally or extend with additional services
- Familiar to many developers and system administrators

Cons:

- Requires managing security patches and uptime
- Increased infrastructure footprint and cost
- More complex to scale and monitor without additional services

---

4. Other Options Considered

- **ECS / EKS:** Powerful container orchestration platforms but add significant complexity for such a minimal service.

- **Azure Functions / Google Cloud Functions:** Viable alternatives, but AWS was selected due to platform familiarity and already having an account.

---

5. Future Enhancements

Several improvements could be on the roadmap:

- **Web UI:** A lightweight interface to allow string updates and visual feedback.
- **Domain name:** Add pretty domain name
- **Cognito SSO:** Secure authentication for internal or multi-tenant access.
- **Monitoring:** CloudWatch dashboards and alerts to track Lambda performance and API Gateway metrics.
- **OpenTofu Modules:** Modularise infrastructure for reuse across environments.
- **Public Documentation:** Generate API specs using Swagger/OpenAPI via API Gateway integrations.

---

6. DevSecOps Considerations

Security and automation are embedded from the outset:

- **GitHub Actions** is used for CI/CD, including linting, formatting, and automated deployments.
- **Dependabot** tracks vulnerable or outdated dependencies.
- Planned improvements could include adding:
  - `tflint` and `tofu fmt` for IaC quality
  - `tfsec` and `trivy` for security scanning
  - `infracost` for cloud cost estimation
- Secrets used in deployments are securely stored in **GitHub Secrets** and referenced in workflows.
- These practices align with **shift-left security** and DevSecOps principles.

---

7. Infrastructure as Code (IaC)

All infrastructure is managed using **OpenTofu**, the community-driven, open-source fork of Terraform maintained by the Linux Foundation.

Why OpenTofu?
Following the relicensing of Terraform under the BSL, OpenTofu ensures vendor neutrality, transparency, and long-term sustainability, while remaining fully compatible with existing Terraform modules and tooling.

Other Considered Tools:

- **CloudFormation:** Verbose and tightly coupled to AWS
- **AWS CDK:** More powerful but adds unnecessary complexity
- **Terraform:** Originally was going to use this, but replaced due to licensing concerns.

Current Setup Includes:

- Lambda, API Gateway, IAM roles, and permissions
- SSM Parameter Store configuration
- Plan and apply split stages in GitHub Actions
- Remote state (S3 + DynamoDB)

---

8. Areas for Future Expansion / Improvement

If I had more time to increase robustness and maturity, the following could be considered:

Authentication & Authorization

- Protect API Gateway with Cognito, IAM, or API keys
- Add throttling and rate limits
- Explore WAF integration for basic threat protection

Observability & Alerting

- Add CloudWatch metrics and custom dashboards
- Alert on 5xx errors, high latency, and failed deployments
- Use structured JSON logs for better analysis

CI/CD Pipeline Maturity

- Harden GitHub Actions with matrix testing and granular workflows
- Add deployment notifications (Slack, email)
- Parameterise deploys by environment
- Automate rollback on failure using OpenTofu state.

Documentation

- Add runbooks and a README with architecture diagrams.
- Use API Gateway's built-in documentation generator.
- Publish ADRs for key architectural choices (e.g., OpenTofu over Terraform)

---

9. Conclusion

AWS Lambda and API Gateway were selected due to their serverless nature, simplicity, and tight integration with other AWS services. **AWS was chosen primarily due to personal familiarity**, allowing for rapid prototyping and confident delivery. In a client or consulting scenario, the choice would be driven by business needs, team skills, compliance, and existing cloud infrastructure.

SSM Parameter Store provided secure configuration management, and **OpenTofu** offered a sustainable, open, and modular foundation for Infrastructure as Code.

GitHub Actions enabled modern CI/CD practices, while DevSecOps principles like dependency scanning and security tooling were embedded into the delivery pipeline.

This project demonstrates how a minimal yet professional-grade architecture can be delivered quickly, with a strong foundation for future scale, security, and automation.