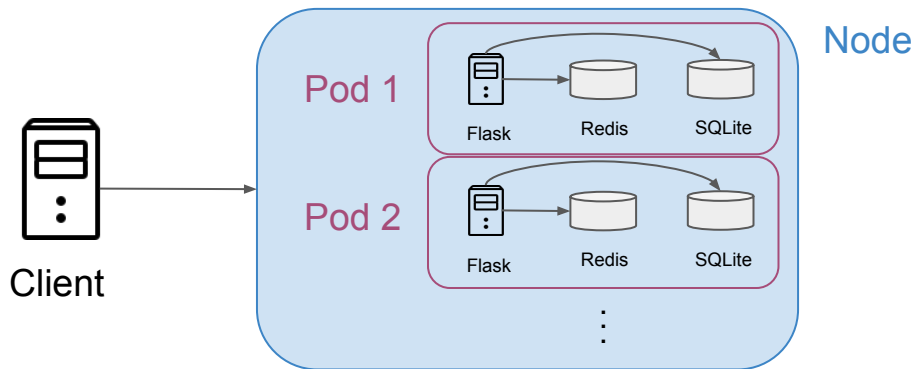


# Scalable Species Info Service

## Project Outline

# Introduction and Architecture

- Prototype for a scalable information service for read-only species data
- Containerized with Docker and deployed locally with Kubernetes (Minikube)
- NodePort service routes requests to one or more species-service pods
  - Each pod contains a Flask app container, a Redis cache container and its own persistent volume for a SQLite database
  - Identical DB seed data across all pods
- Supports horizontal scaling (number of pods) and vertical scaling (CPU/RAM per pod)



# Implemented Features

- Flask API is stateless per request
- **Persistent state** (species data) is identical and preloaded for each pod  
→ No coordination is needed for managing the state
- The Flask app implements a simple **Load Shedding** mechanism to limit the number of requests per minute
- A **Redis Cache** reduces database reads and improves throughput
- **Scaling Script** (.sh) to adjust the Kubernetes resources while the application is running

Repository:

⇒ <https://github.com/serdarakol/ScalabilityEngPrototype/tree/main>

# Demo Functionality

The screenshot shows a VS Code editor window with the following components:

- File Explorer (Left):** Displays the project structure. Folders include `plots/first_experiment`, `src`, `client`, `node_modules`, `load_generator.js`, `package-lock.json`, `package.json`, `data_generator`, `k8s`, `scaler`, `server`, and `.gitignore`. Files include `analyze_log.py`, `docker-compose.yml`, `LICENSE`, `README.md`, and `summary_1st_experi...`.
- Editor (Center):** Shows the `README.md` file. The content includes instructions for applying horizontal scaling, such as running `./src/scaler/script.sh --memory 512Mi --restart` and `./src/scaler/script.sh --cpu 300m --restart`. It also mentions a GitHub repository link: `https://github.com/SerdarAkol/ScalabilityEngPrototype`.
- Terminal (Bottom):** Shows the execution of various commands. The output includes the creation of namespaces, services, and StatefulSets, as well as the application of horizontal scaling. The terminal also shows the execution of `python analyze_log.py` and `chmod +x src/scaler/script.sh`.
- Problems (Bottom Right):** Shows a list of errors, including `registry.k8s.io/coredns/coredns v1.11.3` and `gcr.io/k8s-minikube/storage-provisioner v5`.

# Demo Scalability

ScalabilityEngPrototype

README.md # ScalabilityEngPrototype

Component	Version	Image	Age	Size
registry.k8s.io/kube-scheduler	v1.32.0	c3ff26fb59f3	6 months ago	142MB
registry.k8s.io/kube-proxy	v1.32.0	2f50386e20bf	6 months ago	514kB
registry.k8s.io/etcd	3.5.16-0	7fc9d4aa817a	9 months ago	60.2MB
registry.k8s.io/coredns/coredns	v1.11.3	2f6c962e7b83	11 months ago	29MB
registry.k8s.io/pause	3.10	afb61768cc38	13 months ago	29MB
gcr.io/k8s-minikube/storage-provisioner	v5	ba04bb24b957	4 years ago	29MB

Then create the kubernetes deployment:

```
'''shell
kubectl create -f src/k8s/architecture.yaml -n prototype
'''shell
minikube service species-svc -n prototype --url
```

this will prompt the local url that is routing to the minikube species-svc

eg: ...

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS

(base) - ScalabilityEngPrototype git:(main) x

zsh  
zsh client  
zsh client  
zsh

# Results and Limitations

- We use a StatefulSet with per instance data volumes to ensure each replica can operate independently, for dynamic data we might introduce a scalable shared database
- Caching provides crucial performance gains, a shared cache layer could optimize cache usage even more
- Our simple load shedding mechanism protects the server from overload, for more sophisticated rate limiting we could use adaptive limits
- Automatically scaling based on metrics like CPU usage would help ensure sufficient performance under varying load