

ASDs

Technology mini-hack with LLM. Phase I

Product description

An application continuity service dynamically backs up and restores the state of applications running within the framework on K8s. The service uses the DSL of the framework to determine the relevant application state and its backup policies. Developers of an application are able to backup and restore a specific version from images and relevant state from the service. The application state is stored externally on the given S3-compatible object storage.

Team: Gorelyi Mikhail, Lukashin Daniil, Derezhovskiy Ilya, Sigal Lev

Repo: https://github.com/gorelyi-code/advanced_software_designers

Report: a link to this slides within project repo/doc storage

Stack

Implementation

- Language Python
- API definition OpenAPI
- Connection server for API python gunicorn
- App framework python FastAPI
- Serialization/state format json

Asynchronous interactions (optional)

- Message queue rabbitmq
- Messaging client library celery

Testing tools pytest

Operations

- App initializer systemd
- Code build makefile
- CI/CD pipeline gitlab
- Delivery method docker
- Logging & monitoring prometheus,
loki, grafana

Prompt 1

Template: Create a list of the best {PL} microservice frameworks for working with {what for}. These frameworks have to fit the idea of our project: {aim}

Request: Create a list of the best python microservice frameworks for working with k8s. These frameworks have to fit the idea of our project: Our application continuity service dynamically backs up and restores the state of applications running within the framework on k8s. The service uses the DSL framework to determine the relevant application state and its backup policies. Developers of an application are able to backup and restore a specific version from images and relevant state from the service. The application state is stored externally on the given S3-compatible object storage.

Prompt 1

Result

Сравнение:

Фреймворк	Управление K8s	Создание бэкапов	Простота использования	Документация
Kubernetes Client	✓	✗	Средняя	✓
Kopf	✓	✗	✓	✓
Kubeflow	✓	✗	Сложная	✓
Helm SDK	✓	✗	Средняя	✗
Rancher API	✓	✗	Средняя	✓
Velero SDK	✗	✓	Средняя	✗

<https://gist.github.com/gorelyi-code/30d04ce15ba262dc68bc21e1e23e8aac>

Prompt 2

Template & Request: Choose the 2 best frameworks from your list and explain your choice

Result

The two best frameworks from the list that align closely with your project's requirements are:

1. FastAPI with Kubernetes Operators (Kopf)
2. Nameko

<https://gist.github.com/gorelyi-code/82d4697d90d69f29d9a36c946ba95b05>

Prompt 2

Result

1. FastAPI with Kubernetes Operators (Kopf)

Why This Choice?

FastAPI is a modern, high-performance web framework for building APIs with Python 3.6+ based on standard Python type hints. It's designed for building APIs quickly and efficiently, with automatic interactive documentation.

Kopf (Kubernetes Operator Framework for Python) allows you to write Kubernetes Operators in Python, enabling you to extend Kubernetes functionality by creating custom resources and controllers.

2. Nameko

Why This Choice?

Nameko is a Python microservices framework that allows you to build services that communicate over Remote Procedure Calls (RPC) and events. It provides a simple and declarative way to define services and their dependencies.

<https://gist.github.com/gorelyi-code/82d4697d90d69f29d9a36c946ba95b05>

Prompts 3 & 4

Template: Create a “hello world” code example for the {framework name} framework, which solves the tasks mentioned before.

Request 1: Create a “hello world” code example for the Kopf framework, which solves the tasks mentioned before.

Request 2: Create a “hello world” code example for the Nameko framework, which solves the tasks mentioned before.

Prompt 3

Result

```
import kopf
import boto3
import json
import os

from kubernetes import config

# Load Kubernetes configuration (In-cluster or from default location)
try:
    config.load_incluster_config()
except config.config_exception.ConfigException:
    config.load_kube_config()

# Initialize AWS S3 client (Assuming environment variables are set for credentials)
s3_client = boto3.client(
    's3',
    endpoint_url=os.getenv('S3_ENDPOINT_URL', 'https://s3.amazonaws.com'),
    aws_access_key_id=os.getenv('AWS_ACCESS_KEY_ID'),
    aws_secret_access_key=os.getenv('AWS_SECRET_ACCESS_KEY'),
)

@kopf.on.create('backuppolicies.example.com', 'v1', 'backuppolicies')
def on_create_backup_policy(spec, **kwargs):
    """
    Handler that gets triggered when a new BackupPolicy is created.
    """
    app_name = spec.get('appName')
```

<https://gist.github.com/gorelyi-code/3b36b2378e561a8a0f887c6319f8d6cd>

Prompt 4

Result

```
from nameko.rpc import rpc
import boto3
import os
import yaml
import uuid
from datetime import datetime

class BackupService:
    name = "backup_service"

    # Initialize the S3 client
    def __init__(self):
        self.s3_client = boto3.client(
            's3',
            endpoint_url=os.getenv('S3_ENDPOINT_URL', 'https://s3.amazonaws.com'),
            aws_access_key_id=os.getenv('AWS_ACCESS_KEY_ID'),
            aws_secret_access_key=os.getenv('AWS_SECRET_ACCESS_KEY'),
        )

    @rpc
    def backup_application_state(self, dsl_policy):
        """
        Back up the application state based on the provided DSL policy.
        """
        # Parse the DSL policy (YAML in this example)
        policy = yaml.safe_load(dsl_policy)
        app_name = policy.get('app_name')
        s3_bucket = policy.get('s3_bucket')
```

<https://gist.github.com/gorelyi-code/c6076d58197b54acb5eeaabc1ca4e797>

Choice

Based on our project's requirements, **FastAPI with Kubernetes Operators (Kopf)** is the top choice due to its ability to:

- Directly interact with and extend Kubernetes using Python.
- Provide a high-performance API layer for developers.
- Implement custom logic for backup and restoration as Kubernetes operators.
- Facilitate the creation of a DSL through well-defined APIs.