KEA

Data design

Product description

The product is a platform for deploying, managing, and scaling machine learning models in production. It offers a secure, flexible environment for automating ML tasks like model versioning, routing, and monitoring. With Kubernetes integration and containerization support, it's designed for developers, ML engineers, and enterprises needing scalable, reliable ML infrastructure.

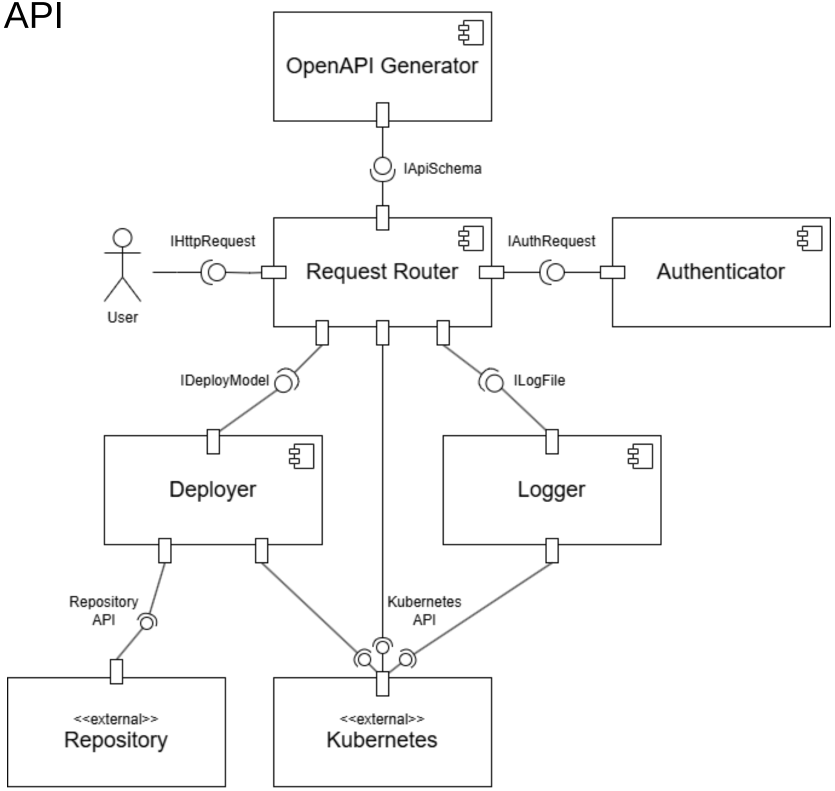
Team K8C: Tsurkan Daniel; Dandamaev Gadji; Tsaturyan Konstantin; Smolkin Mikhail

Project repo: https://github.com/fanglores/Advanced-Software-Design

This report: https://github.com/fanglores/Advanced-Software-Design/blob/master/Practice%20Tasks/Module2/Task_12/Task_12.pdf

System architecture

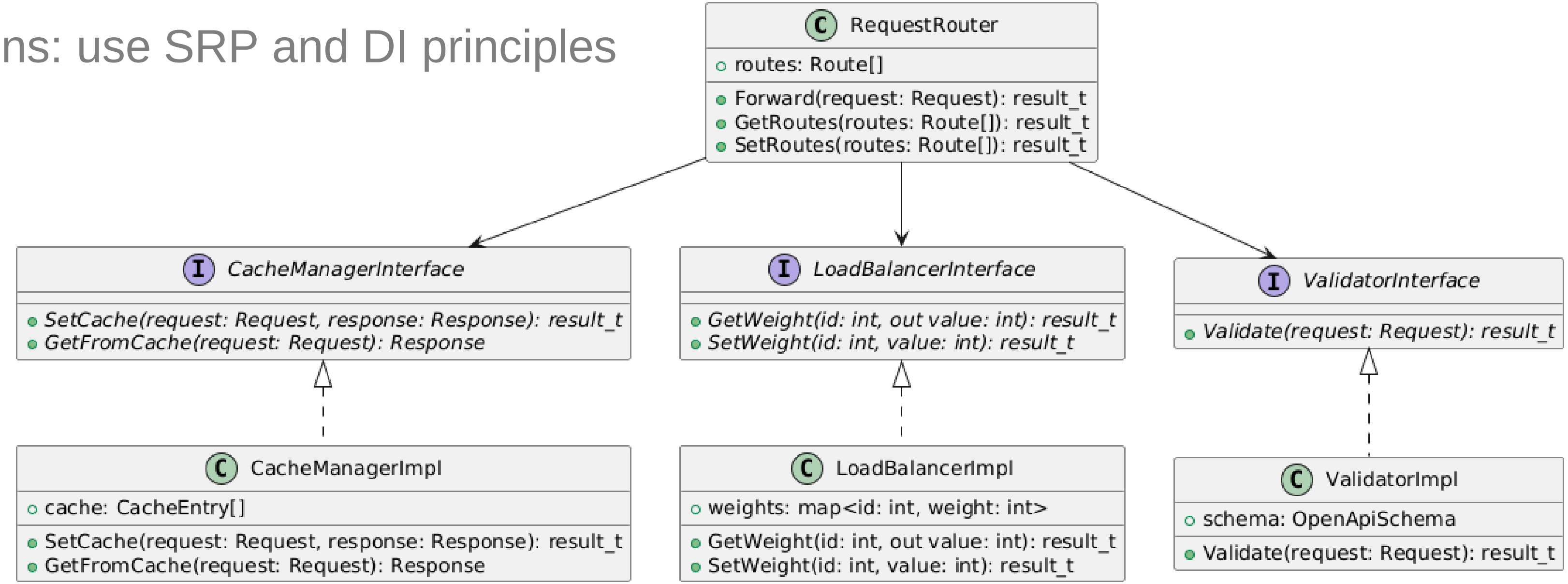
BASE, Microservices, RESTful API


OpenAPI Generator
IApiSchema
IHttpRequest
User
Request Router
IAuthRequest
Authenticator
IDeployModel
ILogFile
Deployer
Logger
Repository
API
Kubernetes
API
<<external>>
Repository
<<external>>
Kubernetes

Design case for RequestRouter

Problems: many functions for one class

Solutions: use SRP and DI principles

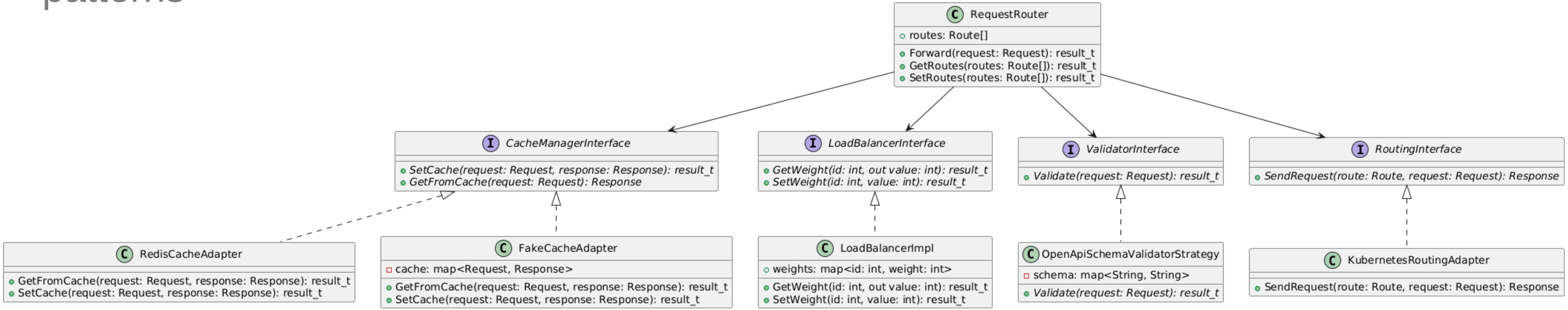


RequestRouter
routes: Route[]
Forward(request: Request): result_t
GetRoutes(routes: Route[]): result_t
SetRoutes(routes: Route[]): result_t
CacheManagerInterface
SetCache(request: Request, response: Response): result_t
GetFromCache(request: Request): Response
LoadBalancerInterface
GetWeight(id: int, out value: int): result_t
SetWeight(id: int, value: int): result_t
ValidatorInterface
Validate(request: Request): result_t
CacheManagerImpl
cache: CacheEntry[]
SetCache(request: Request, response: Response): result_t
GetFromCache(request: Request): Response
LoadBalancerImpl
weights: map<id: int, weight: int>
GetWeight(id: int, out value: int): result_t
SetWeight(id: int, value: int): result_t
ValidatorImpl
schema: OpenApiSchema
Validate(request: Request): result_t

Design case for RequestRouter

Problems: strong dependency on Kubernetes, OpenAPI schemas, cache storage

Solutions: use Strategy and Adapter patterns



RequestRouter
- routes: Route[]
- Forward(request: Request): result_t
- GetRoutes(routes: Route[]): result_t
- SetRoutes(routes: Route[]): result_t

CacheManagerInterface
- *SetCache(request: Request, response: Response): result_t*
- *GetFromCache(request: Request): Response*

LoadBalancerInterface
- *GetWeight(id: int, out value: int): result_t*
- *SetWeight(id: int, value: int): result_t*

ValidatorInterface
- *Validate(request: Request): result_t*

RoutingInterface
- *SendRequest(route: Route, request: Request): Response*

RedisCacheAdapter
- GetFromCache(request: Request, response: Response): result_t
- SetCache(request: Request, response: Response): result_t

FakeCacheAdapter
- cache: map<Request, Response>
- GetFromCache(request: Request, response: Response): result_t
- SetCache(request: Request, response: Response): result_t

LoadBalancerImpl
- weights: map<id: int, weight: int>
- GetWeight(id: int, out value: int): result_t
- SetWeight(id: int, value: int): result_t

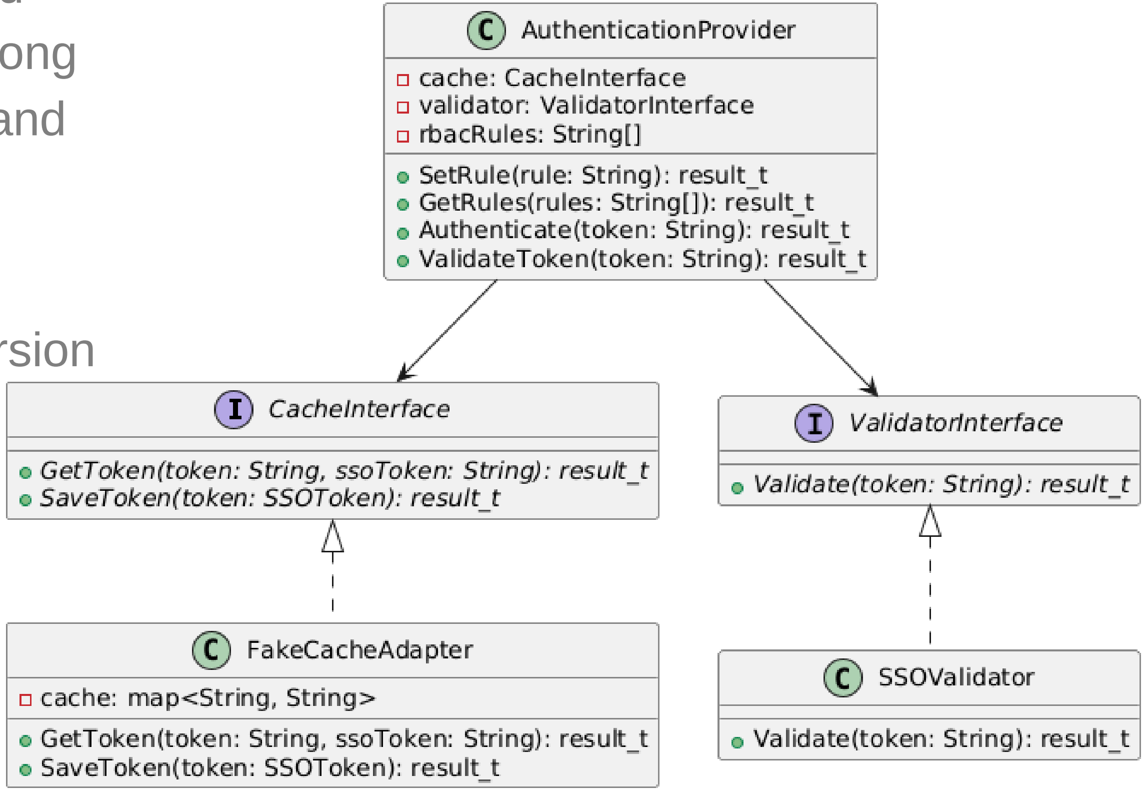
OpenApiSchemaValidatorStrategy
- schema: map<String, String>
- *Validate(request: Request): result_t*

KubernetesRoutingAdapter
- SendRequest(route: Route, request: Request): Response

Design case for Authenticator

Problem: caching, validating and authentication are all in one, strong dependency on cache storage and token type

Solution: use Dependency inversion principle, interface for different tokens and Adapter pattern



AuthenticationProvider

- □ cache: CacheInterface
- □ validator: ValidatorInterface
- □ rbacRules: String[]
- ● SetRule(rule: String): result_t
- ● GetRules(rules: String[]): result_t
- ● Authenticate(token: String): result_t
- ● ValidateToken(token: String): result_t

CacheInterface

- ● *GetToken(token: String, ssoToken: String): result_t*
- ● *SaveToken(token: SSOToken): result_t*

ValidatorInterface

- ● *Validate(token: String): result_t*

FakeCacheAdapter

- □ cache: map<String, String>
- ● GetToken(token: String, ssoToken: String): result_t
- ● SaveToken(token: SSOToken): result_t

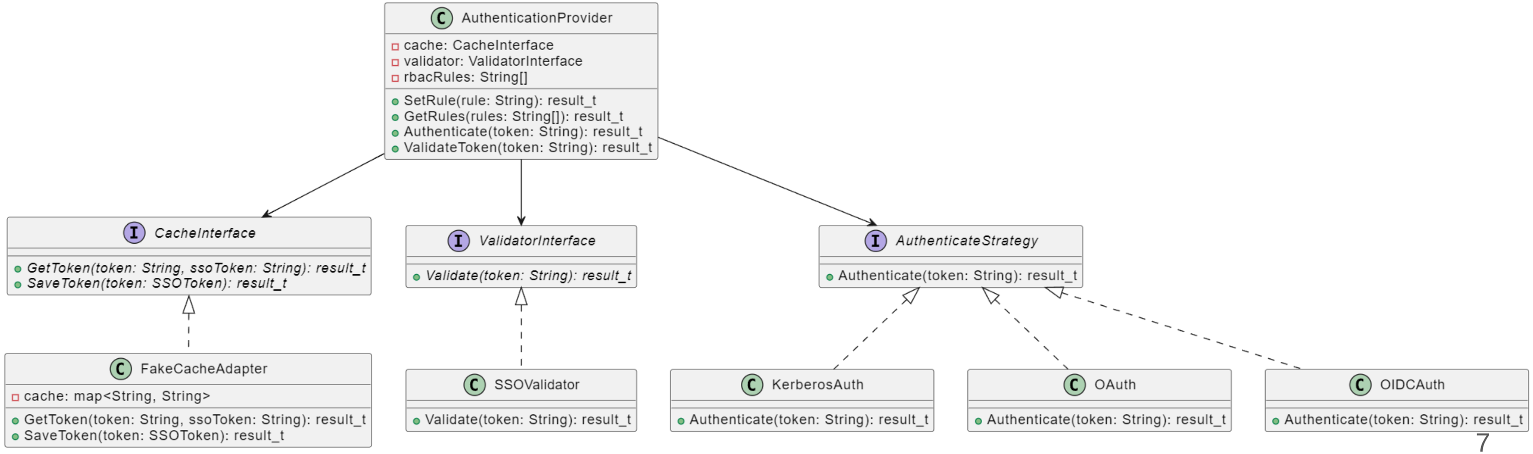
SSOValidator

- ● Validate(token: String): result_t

Design case for Authenticator

Problem: all auth-processes are implemented in AuthenticationProvider

Solution: use Strategy pattern for separating different auth-protocols



AuthenticationProvider
- □ cache: CacheInterface
- □ validator: ValidatorInterface
- □ rbacRules: String[]
- ● SetRule(rule: String): result_t
- ● GetRules(rules: String[]): result_t
- ● Authenticate(token: String): result_t
- ● ValidateToken(token: String): result_t

CacheInterface
- ● *GetToken(token: String, ssoToken: String): result_t*
- ● *SaveToken(token: SSOToken): result_t*

ValidatorInterface
- ● *Validate(token: String): result_t*

AuthenticateStrategy
- ● Authenticate(token: String): result_t

FakeCacheAdapter
- □ cache: map<String, String>
- ● GetToken(token: String, ssoToken: String): result_t
- ● SaveToken(token: SSOToken): result_t

SSOValidator
- ● Validate(token: String): result_t

KerberosAuth
- ● Authenticate(token: String): result_t

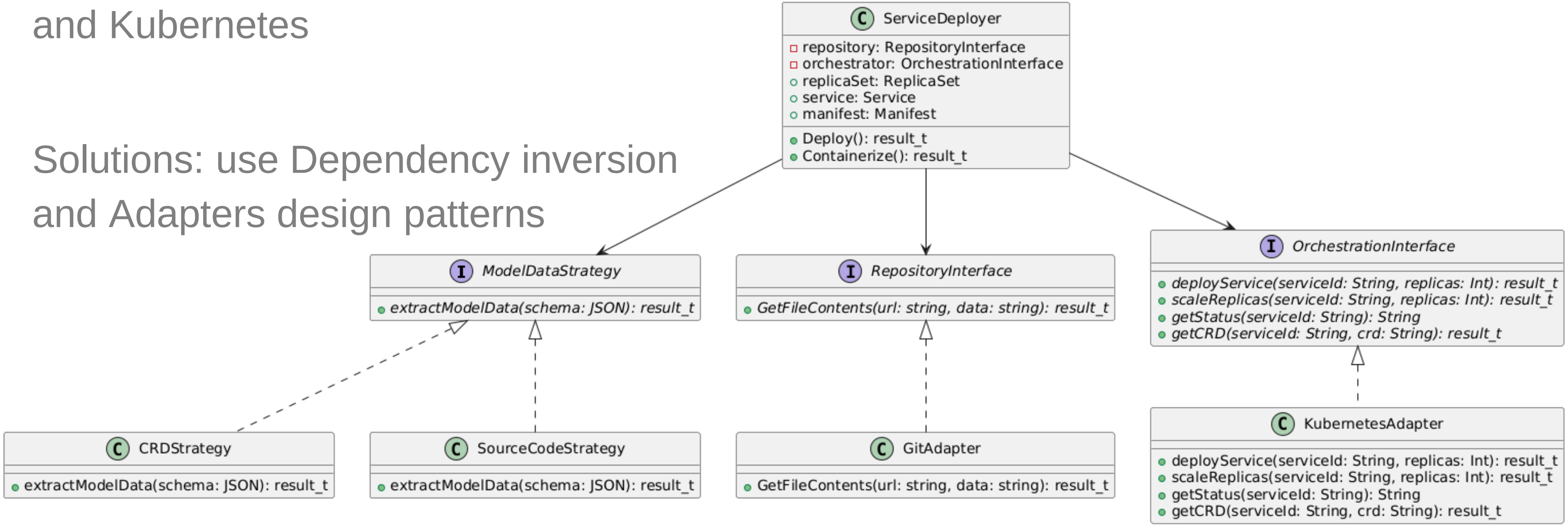
OAuth
- ● Authenticate(token: String): result_t

OIDCAuth
- ● Authenticate(token: String): result_t

Design case of Service Deployer

Problems: need to work with different data (source code from repo or CRD), strong dependency on repo and Kubernetes

Solutions: use Dependency inversion and Adapters design patterns



C ServiceDeployer
- repository: RepositoryInterface
- orchestrator: OrchestrationInterface
- replicaSet: ReplicaSet
- service: Service
- manifest: Manifest
- Deploy(): result_t
- Containerize(): result_t

I *ModelDataStrategy*
- *extractModelData(schema: JSON): result_t*

I *RepositoryInterface*
- *GetFileContents(url: string, data: string): result_t*

I *OrchestrationInterface*
- *deployService(serviceId: String, replicas: Int): result_t*
- *scaleReplicas(serviceId: String, replicas: Int): result_t*
- *getStatus(serviceId: String): String*
- *getCRD(serviceId: String, crd: String): result_t*

C CRDStrategy
- extractModelData(schema: JSON): result_t

C SourceCodeStrategy
- extractModelData(schema: JSON): result_t

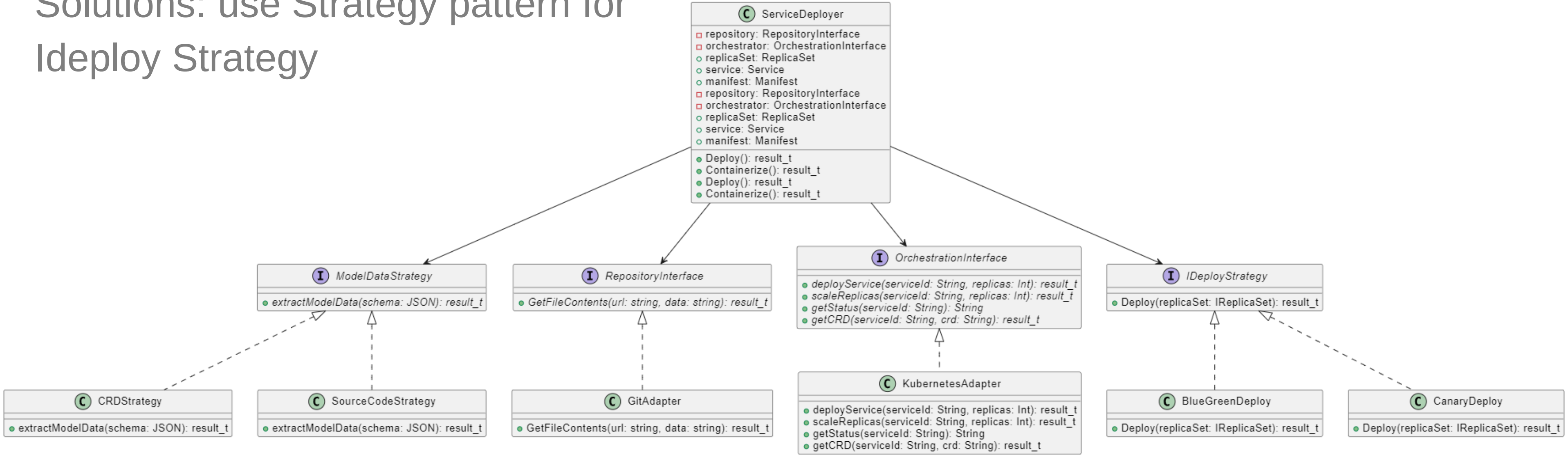
C GitAdapter
- GetFileContents(url: string, data: string): result_t

C KubernetesAdapter
- deployService(serviceId: String, replicas: Int): result_t
- scaleReplicas(serviceId: String, replicas: Int): result_t
- getStatus(serviceId: String): String
- getCRD(serviceId: String, crd: String): result_t

Design case of Service Deployer

Problems: applying for new deploy strategies requires changes in ServiceDeployer

Solutions: use Strategy pattern for Ideploy Strategy


ServiceDeployer
repository: RepositoryInterface
orchestrator: OrchestrationInterface
replicaSet: ReplicaSet
service: Service
manifest: Manifest
repository: RepositoryInterface
orchestrator: OrchestrationInterface
replicaSet: ReplicaSet
service: Service
manifest: Manifest
Deploy(): result_t
Containerize(): result_t
Deploy(): result_t
Containerize(): result_t
ModelDataStrategy
extractModelData(schema: JSON): result_t
RepositoryInterface
GetFileContents(url: string, data: string): result_t
OrchestrationInterface
deployService(serviceId: String, replicas: Int): result_t
scaleReplicas(serviceId: String, replicas: Int): result_t
getStatus(serviceId: String): String
getCRD(serviceId: String, crd: String): result_t
IDeployStrategy
Deploy(replicaSet: IReplicaSet): result_t
CRDStrategy
extractModelData(schema: JSON): result_t
SourceCodeStrategy
extractModelData(schema: JSON): result_t
GitAdapter
GetFileContents(url: string, data: string): result_t
KubernetesAdapter
deployService(serviceId: String, replicas: Int): result_t
scaleReplicas(serviceId: String, replicas: Int): result_t
getStatus(serviceId: String): String
getCRD(serviceId: String, crd: String): result_t
BlueGreenDeploy
Deploy(replicaSet: IReplicaSet): result_t
CanaryDeploy
Deploy(replicaSet: IReplicaSet): result_t