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.

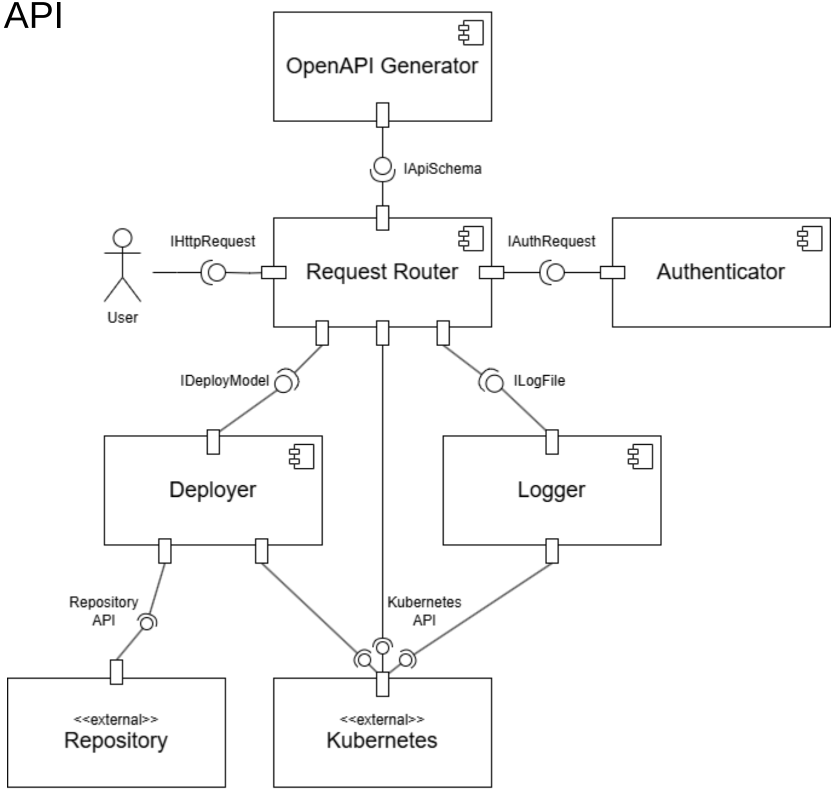
Team K8C: Tsurkan Daniel; Dandamaev Gadji; Tsaturyan Konstantin; Smolkin Mikhail

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

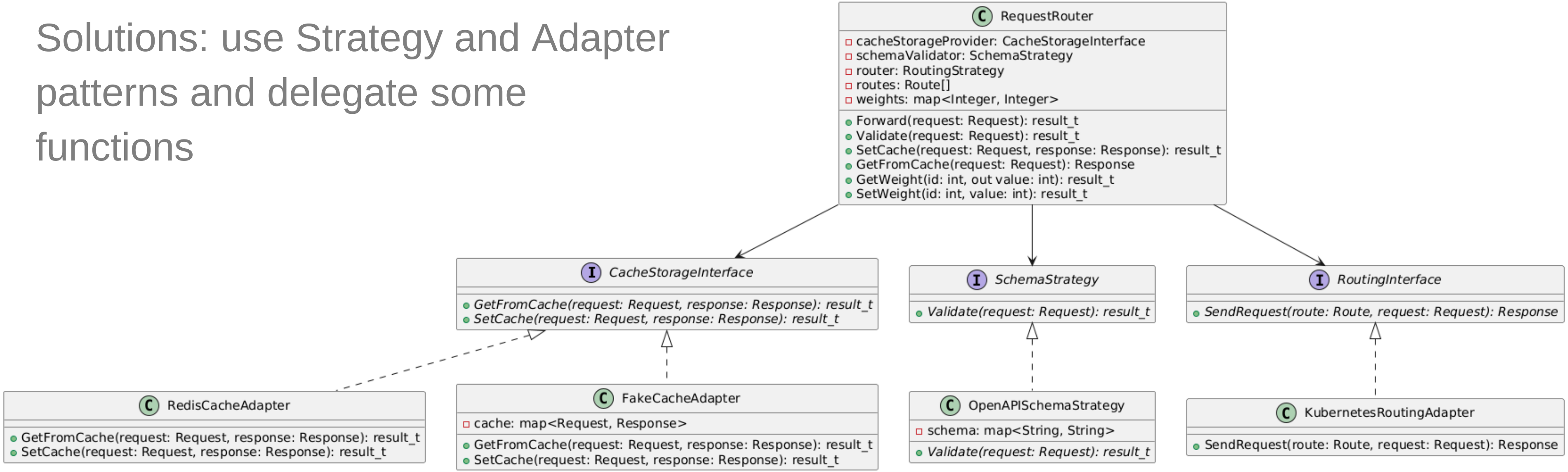
BASE, Microservices, RESTful API



Design case for RequestRouter

Problems: strong dependency on Kubernetes, OpenAPI schemas, many functions for one class

Solutions: use Strategy and Adapter patterns and delegate some functions



C RequestRouter

- cacheStorageProvider: CacheStorageInterface
- schemaValidator: SchemaStrategy
- router: RoutingStrategy
- routes: Route[]
- weights: map<Integer, Integer>

- Forward(request: Request): result_t
- Validate(request: Request): result_t
- SetCache(request: Request, response: Response): result_t
- GetFromCache(request: Request): Response
- GetWeight(id: int, out value: int): result_t
- SetWeight(id: int, value: int): result_t

I *CacheStorageInterface*

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

I *SchemaStrategy*

- *Validate(request: Request): result_t*

I *RoutingInterface*

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

C RedisCacheAdapter

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

C FakeCacheAdapter

- cache: map<Request, Response>

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

C OpenAPISchemaStrategy

- schema: map<String, String>

- *Validate(request: Request): result_t*

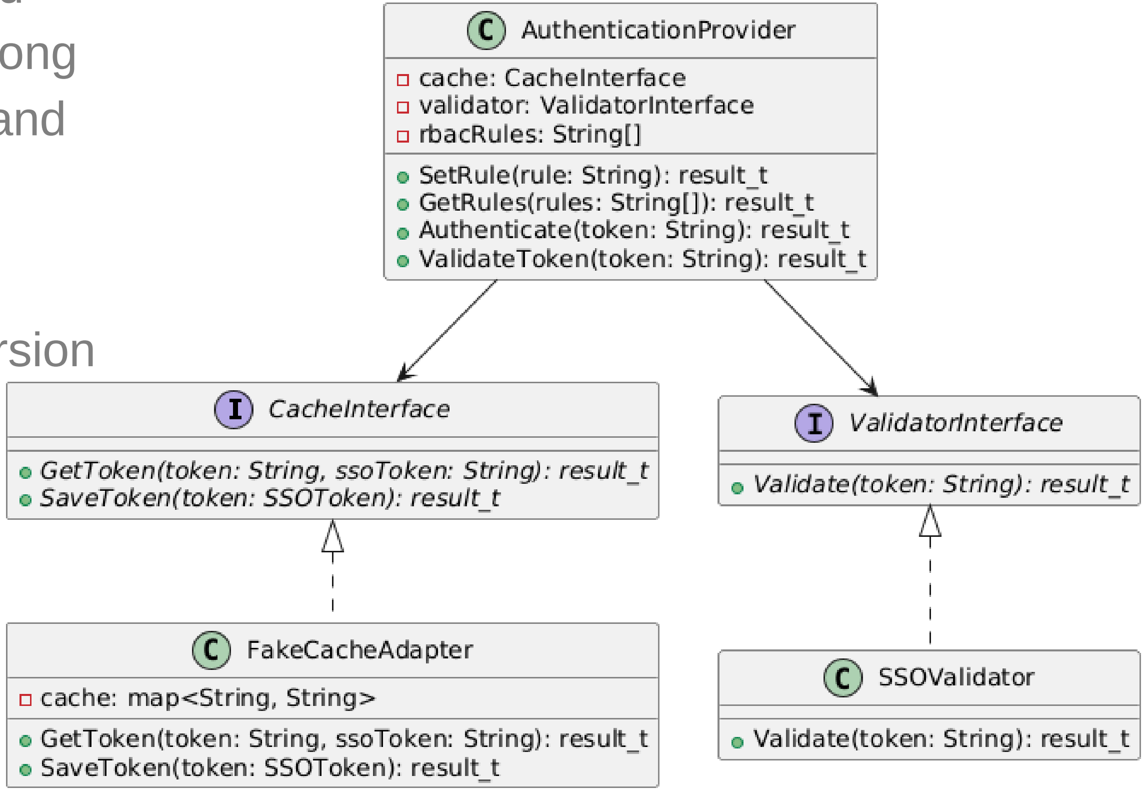
C KubernetesRoutingAdapter

- 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



C 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

I *CacheInterface*

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

I *ValidatorInterface*

+ *Validate(token: String): result_t*

C FakeCacheAdapter

- cache: map<String, String>

+ GetToken(token: String, ssoToken: String): result_t
+ SaveToken(token: SSOToken): result_t

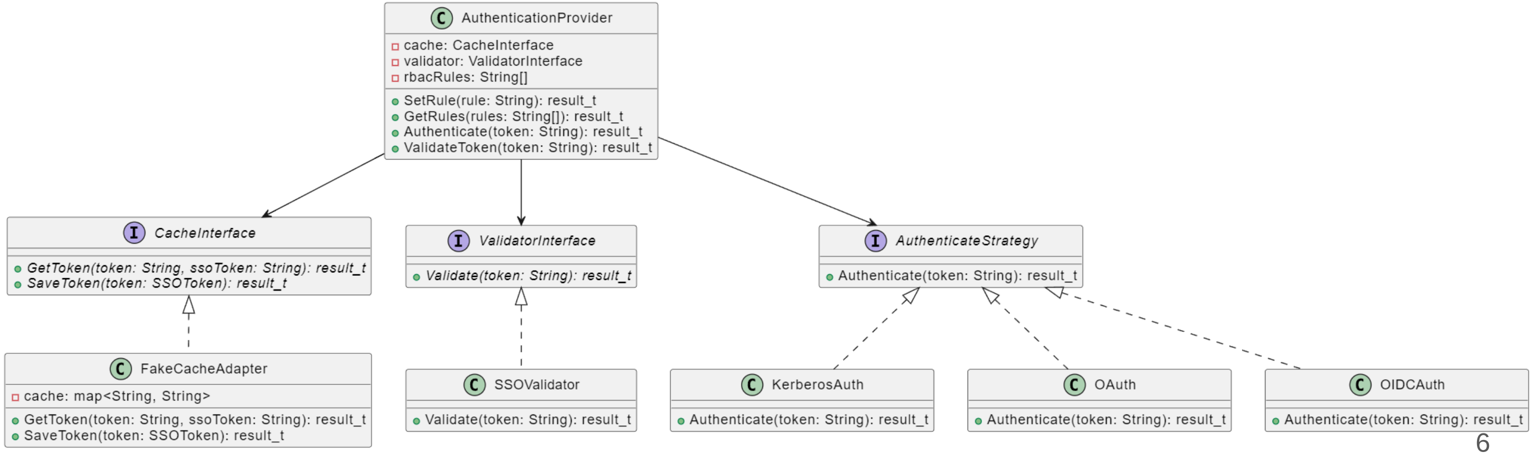
C SSOValidator

+ 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



C 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

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

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

I *AuthenticateStrategy*
- Authenticate(token: String): result_t

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

C SSOValidator
- Validate(token: String): result_t

C KerberosAuth
- Authenticate(token: String): result_t

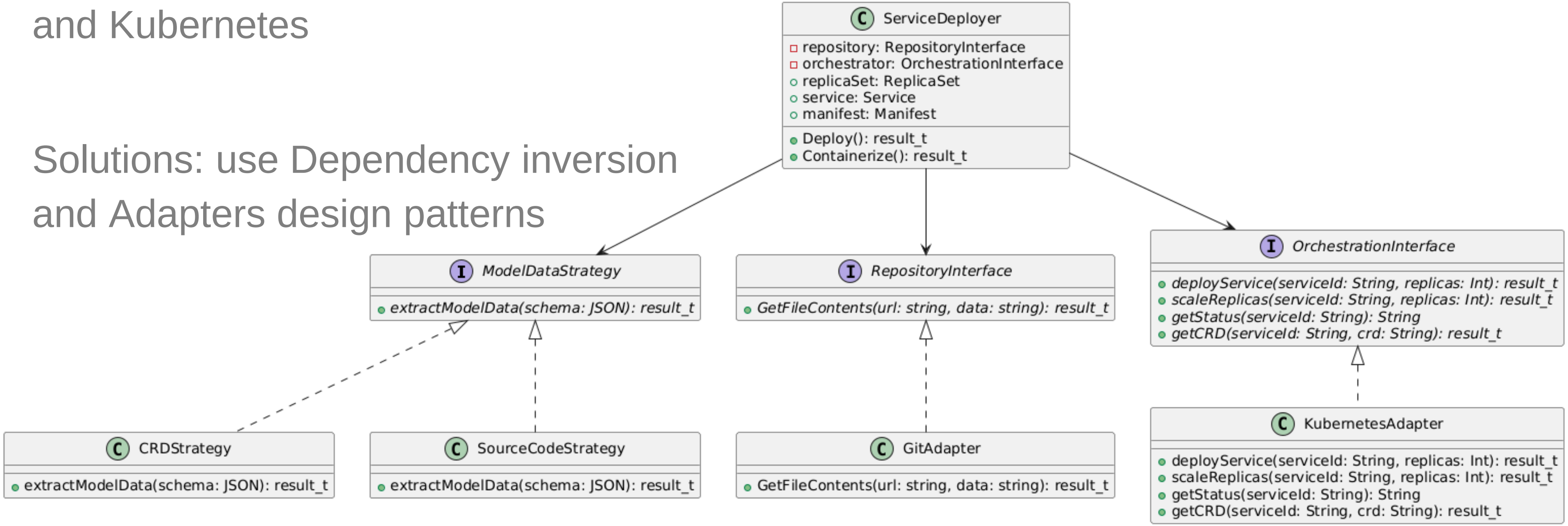
C OAuth
- Authenticate(token: String): result_t

C 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

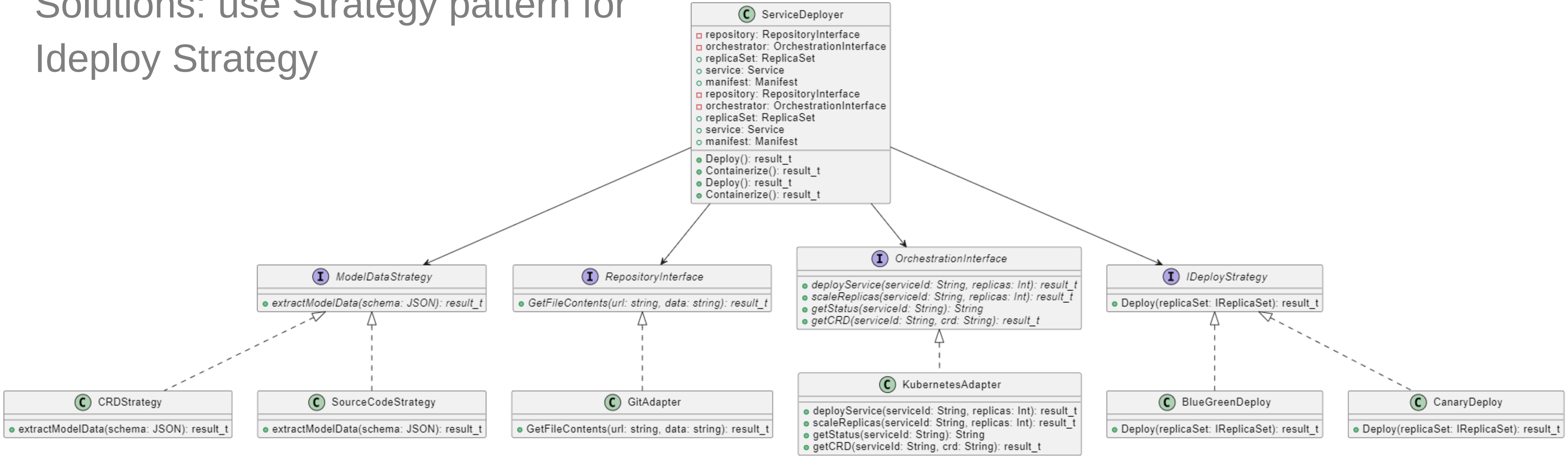
C GitAdapter
- ● GetFileContents(url: string, data: string): result_t

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



C 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

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*

I *IDeployStrategy*
- Deploy(replicaSet: IReplicaSet): 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

C BlueGreenDeploy
- Deploy(replicaSet: IReplicaSet): result_t

C CanaryDeploy
- Deploy(replicaSet: IReplicaSet): result_t