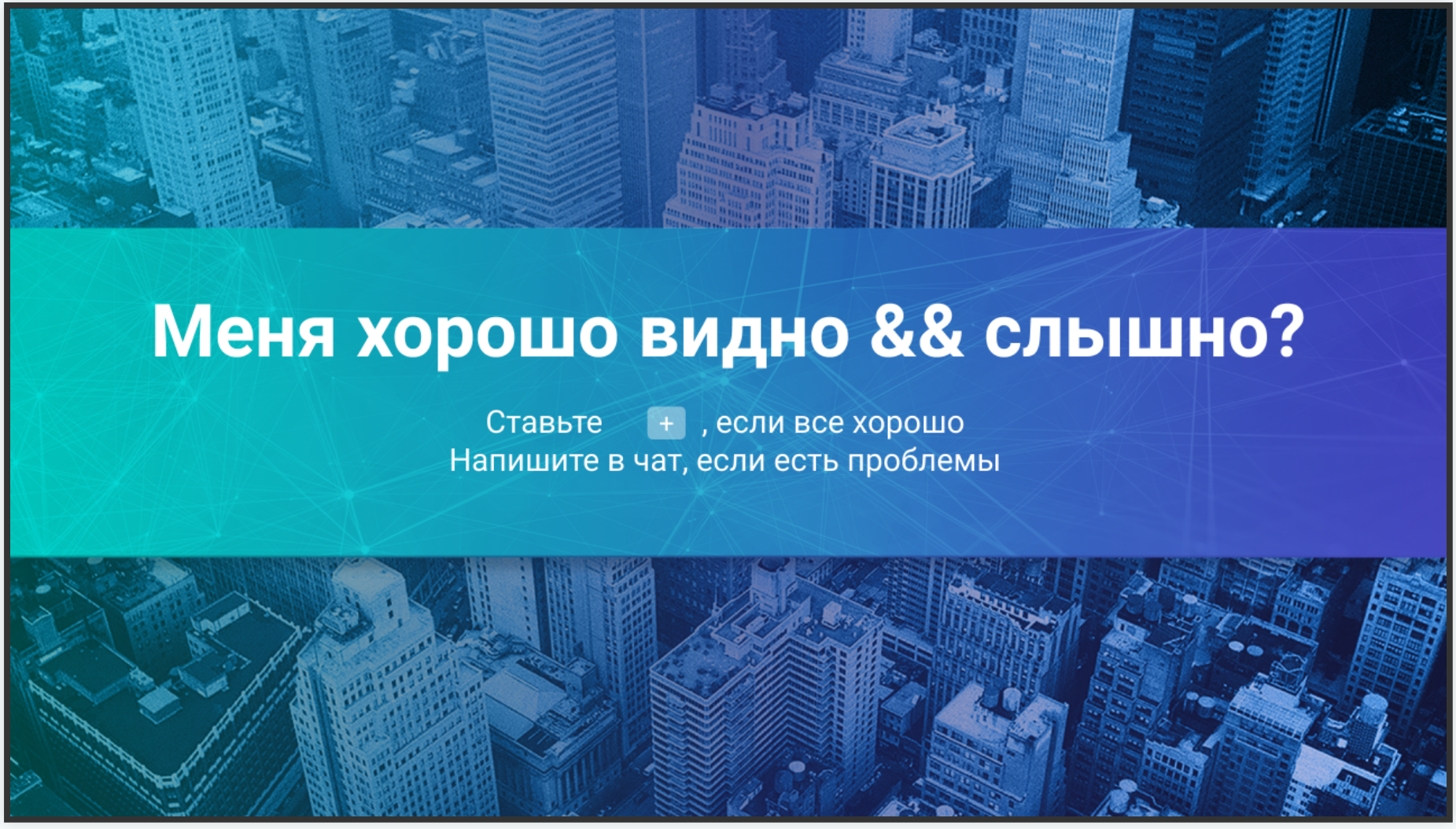

OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование


Меня хорошо видно && слышно?
Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

НЕ ЗАБЫТЬ ВКЛЮЧИТЬ ЗАПИСЬ!!!

Работа с секретами. Hashicorp Vault.

План

- распространенные практики работы с секретами
- ansible-vault
- gitlab variables
- gcloud secrets
- работа с секретами в kubernetes
- Хранение секретов в vault
- Использование vault в k8s

Практики работы с секретами

- конфиги, файлы
- env
- git
- хранилище CI (gitlab)
- системы облачных провайдеров
- специализированные системы (hashicorp vault)

Генерация паролей

```
export EPASS=$(cat /dev/urandom | \
tr -dc 'a-zA-Z0-9' | fold -w 32 | head -n 1)
export ECOLLECTION=EXNESS_$(cat /dev/urandom \
| tr -dc 'a-zA-Z0-9' | fold -w 4 | head -n 1)
helm repo add bitnami https://charts.bitnami.com/bitnami
envsubst <  values.yaml | helm  install --wait  --timeout 60s \
otus-mongodb bitnami/mongodb -f -
echo
echo "created user test-user with password $EPASS"
echo "created collection $ECOLLECTION"
```

Ansible-vault

- для безопасного хранения секретов в ролях и плейбуках можно (и нужно) использовать ansible-vault
- с помощью него можно шифровать целиком файлы или отдельные переменные

Ansible-vault: шифрование файлов

```
ansible-vault create secrets.yml
ansible-vault view secrets.yml
ansible-vault edit secrets.yml
ansible-vault rekey secrets.yml
```

- ansible.cfg
  - vault_password_file = .vault
- ansible-playbook
  - `--vault-password-file`
  - `--ask-vault-password`

Ansible-vault: шифрование строк

```
ansible-vault encrypt_string <password_source> \
'<string_to_encrypt>' --name '<string_name_of_variable>'
# пример
ansible-vault encrypt_string  'llsJSHKSH#12' --name 'otus_secret'
# шифрование с stdin
echo -n 'letmein' | ansible-vault encrypt_string \
 --stdin-name 'test_db_password'

# чтение зашифрованных переменных
ansible localhost -m debug -a var="otus_secret" \
-e "@test_vars.yml"
```

Anaible-vault. Несколько паролей

- vaultfile

```
dev my_dev_pass
test my_test_pass
prod my_prod_pass
```

- выбор пароля `--vault-id dev@vaultfile`
- перешифрование `ansible-vault rekey --vault-id preprod1@ppold --new-vault-id preprod2@prompt foo.yml bar.yml baz.yml`

GitLab variables

- Встроенные
- Пользовательские
- Секретные

Хранение секретов в облаке (GCP)

```
gcloud secrets create SECRET [--data-file=PATH] \
[--labels=[KEY=VALUE,...]] \
[--locations=[LOCATION,...]] \
[--replication-policy=POLICY] \
[GCLOUD_WIDE_FLAG ...]
```

- gcloud secrets list/desribe/update/delete
- Вариант использования

Хранение секретов в k8s

- Создание секрета

```
kubectl create secret generic dev-db-secret \
--from-literal=username=devuser \
--from-literal=password='S!B#d#zDsb'
```

- Создание секрета из файла

```
kubectl create secret generic db-user-pass \
--from-file=./username.txt \
--from-file=./password.txt
```

- Создание секрета из yaml'a
  - для записи в yaml используем base64
- Хранение сертификатов

```
kubectl create secret tls vault-certs \
--cert=vault.crt --key=vault_gke.key
```

Использование секретов в k8s

- Монтирование секрета через volumeMounts в файл
- Мэппинг в переменные окружения пода

```
env:
- name: WORDPRESS_DB_PASSWORD
  valueFrom:
    secretKeyRef:
      name: cloudsql-db-credentials
      key: password
```

Volume Mount

```
    containers:
        ....
        volumeMounts:
          - name: cloudsql-instance-credentials
            mountPath: /secrets/cloudsql
            readOnly: true
     volumes:
       - name: cloudsql-instance-credentials
         secret:
           secretName: cloudsql-instance-credentials
```

Шифрование хранилища секретов

- По умолчанию секреты сохраняются в etcd в нешифрованном виде
- Но есть возможность зашифровать с использованием одного из провайдеров
  - aescbc
  - secretbox
  - aesgcm
  - kms

Остались вопросы про секреты?

Hashicorp vault

Особеннности

- REST, JSON
- Безопасно хранит и управляет ключами.
- хранилища: file, consul, etcd, mysql, mongodb ...
- кластеризуется (в том числе в k8s)
- удобная система политик
- наличие API для взаимодействия из приложений напрямую
- различные варианты аутентификации: userpass, tls, токены, внешние API ...

Проблемы и решения от Vault

Проблемы	Возможности vault
Хранение секретов везде.	Одно общее хранилище.
Хранение в открытом виде.	Встроенное шифрование, включая транзитное
Сложности с динамическими секретами	Встроенная возможность динамической генерации

Demo - vault UI

- kubectl get svc vault-ui

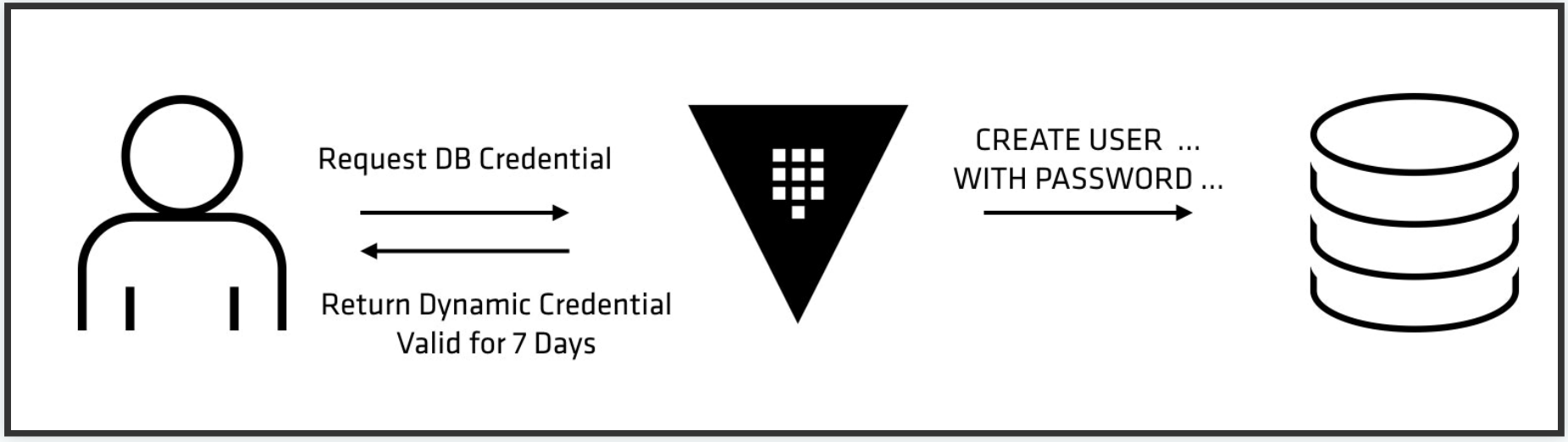
Vault secrets engines

- Компоненты для хранения, генерации и шифрования секретов
  - проще воспринимать как функцию для реализации этих методов
- Может хранить локально, может обращаться на внешний API
- Обращение к компоненту происходит через `path`
- Жизненный цикл secret engines
  - enable/disable
  - move
  - tune
- Помощь по конкретному энжину `vault path-help`

Динамические секреты

- Создаются по требованию
- ограниченный доступ согласно роли
- могут быть ограничены сроком жизни
- могут быть отозваны
- доступ к секрету логгируется (включен аудит)

Динамические секреты



Vault leases

- Метаинформация о динамических секретах и токенах
  - содержит ttl, возможность обновления итд
- При чтении выдается LeaseId, через который можно
  - `vault renew my-lease-id 3600`
  - `vault revoke my-lease-id`

Vault wrapping

- Доступ к секрету через временный токен
  - при чтении секрета создается временный токет
  - у токена ограничен TTL
  - передается токен
  - по токену читается секрет

Vault policies

- Задаются файлами правил
  - `vault policy write my_policy my_policy.hcl`
  - my-policy.hcl

```
path "secret/app/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
path "secret/db/*" {
   policy = "read"
}
```

Транзитное шифрование

```
vault secrets enable transit
vault write -f transit/keys/otus
vault write transit/encrypt/otus plaintext=$(base64 <<< "Hell, ki
vault write transit/decrypt/otus ciphertext=<cipher> | base64 -d
```

Policy capabilities

- create - позволяет создавать секреты по заданному пути
- read - разрашает чтение
- update - разврешает обновление существующих секретов
- delete - разрешает удаление существующих секретов
- list - просмотр списка секретов
- sudo - позволяет доступ к root-protected секретам
- deny - запрещает доступ

Vault audit

```
# file
vault audit enable file file_path=/var/log/vault_audit.log
# stdout
vault audit enable file file_path=stdout
# syslog
vault audit enable syslog tag="vault" facility="AUTH"
# tcp/udp/unix socket
vault audit enable syslog tag="vault" facility="AUTH" format="jso
```

Deploy в kubernetes

- Рекомендуемый backend - consul
- (consul-helm)[https://github.com/hashicorp/consul-helm]
- (vault-helm)[https://github.com/hashicorp/vault-helm]

На что обратить внимание при деплое

- gossip-encryption-key в consul
- HA
- HTTPS
- включить UI

CSR

vi vault_gke_csr.cnf

```
[ req ]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
req_extensions = v3_ext
[ dn ]
commonName = localhost
stateOrProvinceName = Moscow
countryName = RU
emailAddress = lucky@perflabs.org
organizationName = Perflabs
organizationalUnitName = Development
[ v3_ext ]
basicConstraints = CA:FALSE
keyUsage = keyEncipherment,dataEncipherment
```

Создаем и подписываем сертификаты для HTTPS

- create a vault key
  - `openssl genrsa -out vault_gke.key 4096`
- create a request
  - `openssl req -config vault_gke_csr.cnf -new -key vault_gke.key -nodes -out vault.csr`

create k8s csr resource

vi vault_csr.yml

```
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: vaultcsr
spec:
  groups:
  - system:authenticated
  request: ${BASE64_CSR}
  usages:
  - digital signature
  - key encipherment
  - server auth
```

apply to k8s

- sign with k8s

```
export BASE64_CSR=$(cat ./vault.csr | base64 | tr -d '\n')
cat vault_csr.yml | envsubst | kubectl apply -f -
kubectl get csr
kubectl certificate approve vaultcsr
```

- get signed cert

```
kubectl get csr vaultcsr -o jsonpath='{.status.certificate}'  | base64 --decode >
kubectl create secret tls vault-certs --cert=vault.crt --key=vault_gke.key
```

Инициализация и открытие хранилища

- После установки требуется инициализация хранилища
  - `vault operator init --key-shares=1 --key-threshold=1`
- После инициализации отдает ключи
  - root'a
  - unseal ключ
- По дефолту работает через HTTP
  - необходимо включить https в конфиге

Смена токена root

```
## change root token
```bash
vault operator generate-root -init
vault operator generate-root

vault operator generate-root -decode=<encoded token> -otp=<OTP>
```
```

Vault аутентификация

- Документация
- Поддерживает несколько auth методов:
  - userpass, token, tls, kubernetes, pki
- По умолчанию каждый тип авторизации хранится по пути `auth/<type>`
- Для использования метода его необходимо включить

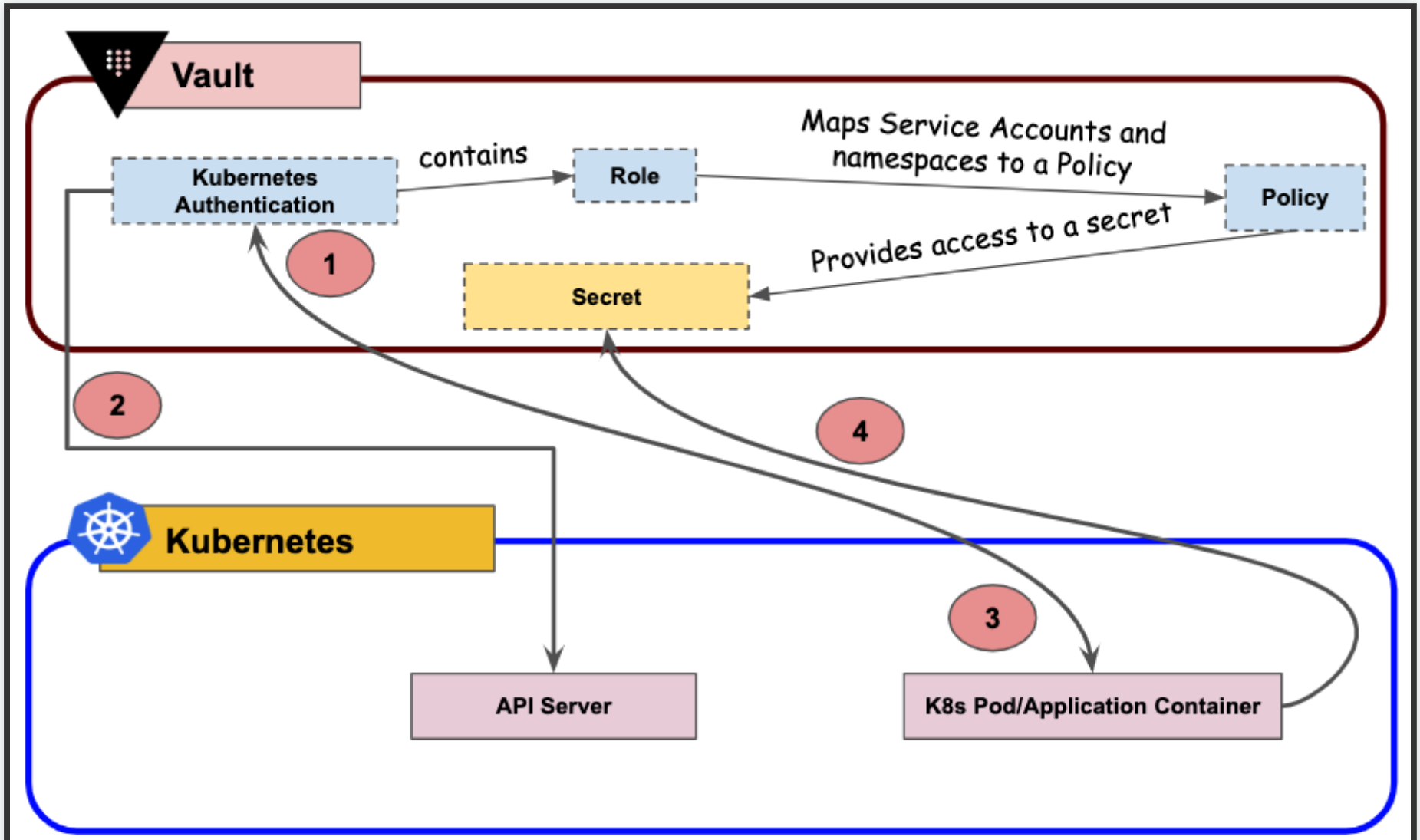
```
vault auth enable userpass
vault write auth/userpass/users/otus \
  password=otusPass \
  policies=otus
vault list auth/userpass/users
vault read auth/userpass/users/otus
```

Key-value secret engine

- Документация

```
vault secrets enable -path=internal kv-v2
vault kv put internal/database/config username="db-readonly-username" password="
vault kv get internal/database/config
```

Kubernetes auth



1. Pod (application supplied JWT with Service Account and Namespace)

Kubernetes auth

```
# включили аутентификацию через k8s
vault auth enable kubernetes

# прописали куда обращаться за api
vault write auth/kubernetes/config \
token_reviewer_jwt="$SA_JWT_TOKEN" \
kubernetes_host="$K8S_HOST" \
kubernetes_ca_cert="$SA_CA_CRT"

# создаем роль с привязкой к сервис аккоунту
vault write auth/kubernetes/role/demo \
    bound_service_account_names=vault-auth \
    bound_service_account_namespaces=default \
    policies=default \
    ttl=1h
```

Kubernetes Service Account

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
- kind: ServiceAccount
  name: vault-auth
  namespace: default
```

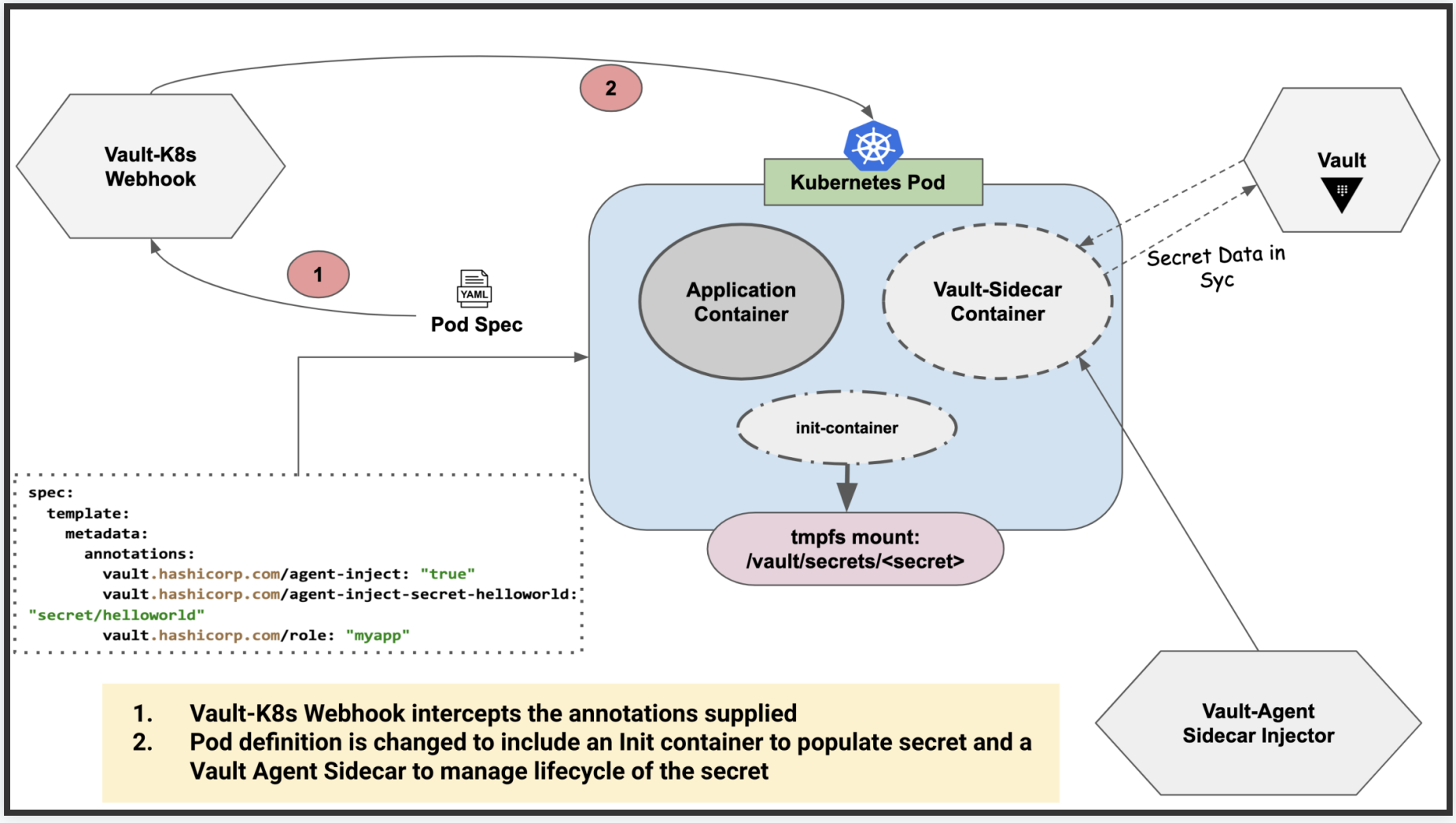
Vault agent

Клиентский демон со следующим функционалом

- Auto-auth
  - автоматическая аутентификация и автообновление токена
- Caching
  - кэширование токенов
- Templating
  - рендер шаблонов с автоподстановкой новых секретов и возможностями вызова команд

Vault agent

Vault inject



Пример инжектора

- patch-inject-secrets.yml

```
spec:
template:
  metadata:
    annotations:
      vault.hashicorp.com/agent-inject: "true"
      vault.hashicorp.com/role: "internal-app"
      vault.hashicorp.com/agent-inject-secret-database-config.txt: "internal/data
```

Инжектор с конфигмэпом

- patch-basic-annotation.yaml

```
spec:
template:
  metadata:
    annotations:
      vault.hashicorp.com/agent-inject: "true"
      vault.hashicorp.com/agent-configmap: vault-injector-configmap
```

- `kubectl patch deployment wordpress --patch "$(cat patch-basic-annotations.yaml)"`

{% include "slides/OTUS_CI-CD/Common/thanks.md.nj" %}