

Hack your Kubernetes controller in Bash in 10 minutes!

<https://github.com/maelvls/hack-your-controller-in-bash>



Code and live
slides!

Antoine
Le Squéren

Maël
Valais


Hack your Kubernetes controller in Bash in 10 minutes!

A story about Vault, external-secrets, slow scaffold run, and how a one-liner Bash controller did the trick.



Antoine Le Squéren,
DevOps Engineer



 [in/antoine-le-squéren-8495a774](https://www.linkedin.com/in/antoine-le-squéren-8495a774)

"I improve the developer experience at OneStock by providing an efficient development environment."



Maël Valais,
Software Engineer



 [@maelvlis](https://twitter.com/maelvlis)  [in/maelvalais](https://www.linkedin.com/in/maelvalais)

"I maintain the cert-manager project. I aim to build the best Let's Encrypt experience on Kubernetes."

Speaker notes

Hi everyone,

Few months ago, I met Mael at a coworking space at Toulouse, in the south of France.

We quickly got along well and we started to speak about our jobs problems during coffee breaks.

I told him I was struggling with a kubernetes problem, it involved Vault and the External Secret Operator.

He told me, alright, I might be able to help you, show me the code.

So we sat in front of my computer and a moment later, he told me, you should write a kubernetes controller !

Ah. Are you serious ? Because it sounds really complicated to me.

You know, I don't contribute to the kubernetes project.

Then he wrote few bash commands and we had a functional prototype which solved my problem nicely.

He made me realised, it's not that hard to write kubernetes controller.

Today we want to share this story with you, because we think that writing small kubernetes controller is not that complicated and it can solve real problems.

I'm Antoine Le Squéren, a devops engineer at OneStock

My goal is to improve developer experience by providing a nice development environment.

I'm Mael, I work at Jetstack. Jetstack is the company behind cert-manager, and cert-manager is a CNCF Sandbox project for getting Let's Encrypt certificates in your Kubernetes cluster.

Part of my job is to maintain cert-manager and I focus on keeping TLS and X.509 boring.

This presentation is about a story.

It starts with Antoine's story at OneStock, and how secrets were slowing down developers.

It continues with our common story, after we met, and how a simple Bash command that turned out to be a controller ended up solving that speed problem.

OneStock



Speaker notes

Yes, so I'm going to briefly speak about OneStock and what problem we want to solve.

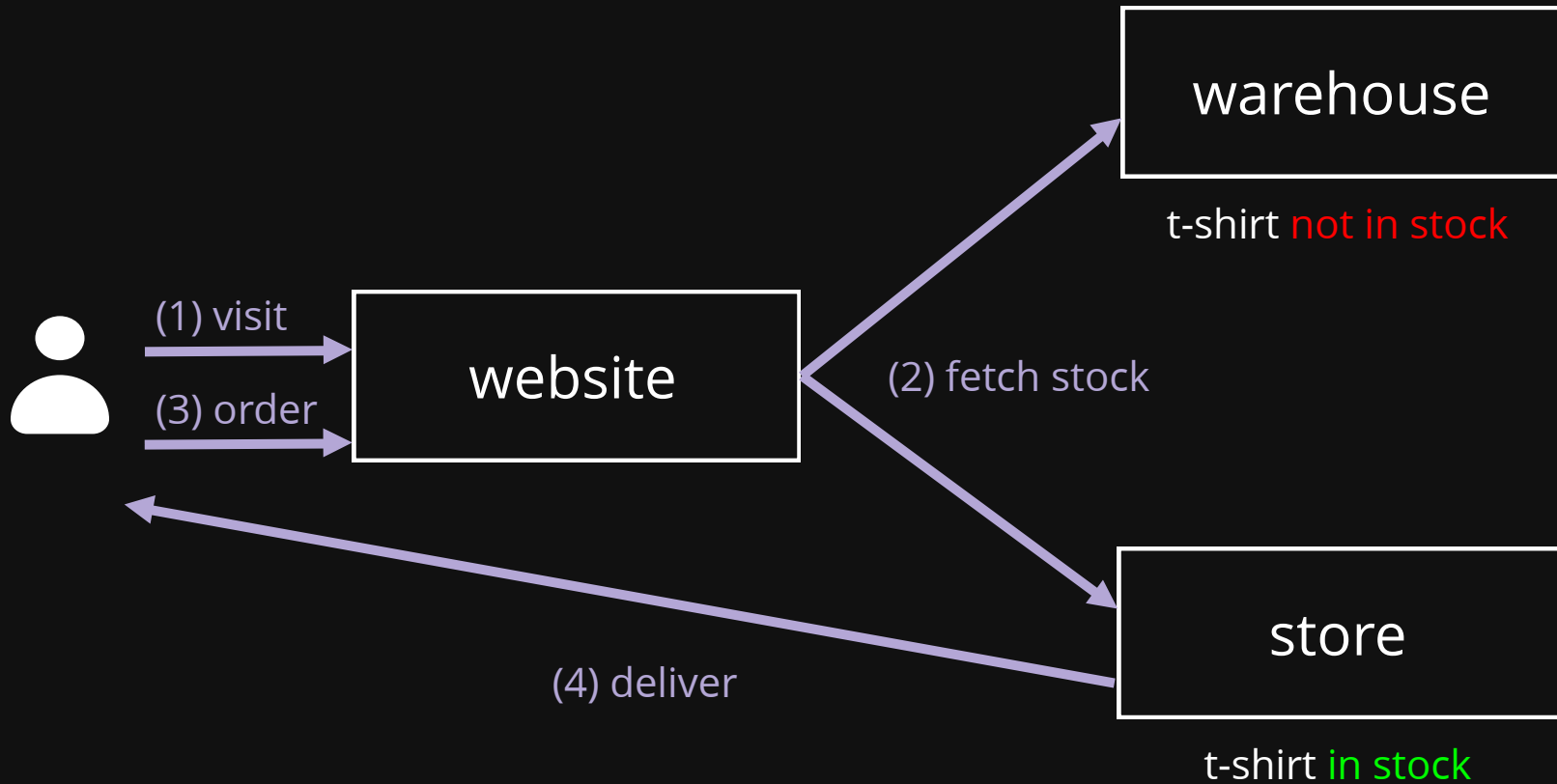
We sell an SAS solution to online retailers for unifying warehouse and store stocks.

Imagine you want to buy a tshirt on Intersport.

You visit the website, the warehouse have that tshirt in stock so you can order it.

The warehouse prepares your parcel and send it to you.

OneStock



Speaker notes

But what happens if the warehouse does not have stock ?

Well without OneStock, you will see an out of stock.

But, it's too bad because store may have you product in the shelves.

But with OneStock, stores and warehouse stocks are unified.

Now you see that your tshirt is in stock and you can order it.

You order will be sent to all stores that can fulfill it and store sellers will be able to claim your order.

The store seller will pickup you product, prepare the parcel and ship it to you.

It's completely transparent on your side.

This is a basic use case to avoid out of stock, but we handle many more.

OneStock has grown quickly over the last years.

More than 10 000 stores use our UI daily and

10% of europeans have already used our service without knowing us.

We are now more than 40 developers.

Secrets at OneStock

Internal secrets

- OneStock API
- Postgre
- MongoDB
- ElasticSearch
- ...

10 secrets

3 environments
(dev, staging, prod)

External secrets:

- Stock API
- SFTP
- Mailjet
- Gmaps
- ...

5 secrets

60 clients
=> 300 secrets

Speaker notes

At OneStock, we handle a lot of secrets as many companies do.

We have 2 kinds of secrets,

On one side, there are internal secrets for our stack credentials.

It includes our databases and api login/passwords used by our services.

These are set for each environment.

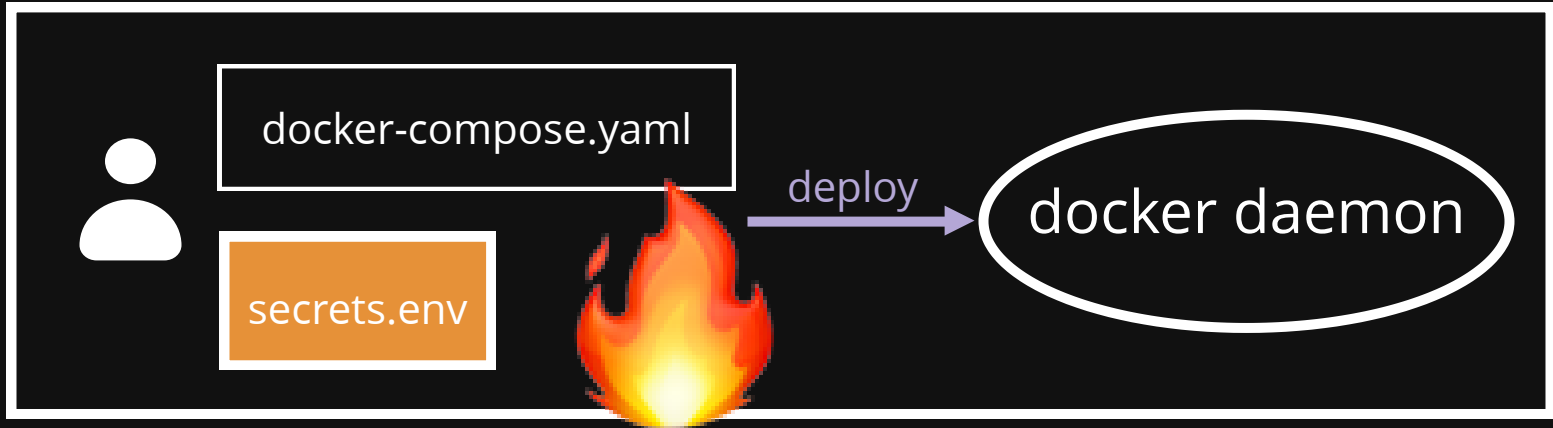
On the other side, there are external secrets.

They contain credentials to connect to external services such as the stock API of our clients.

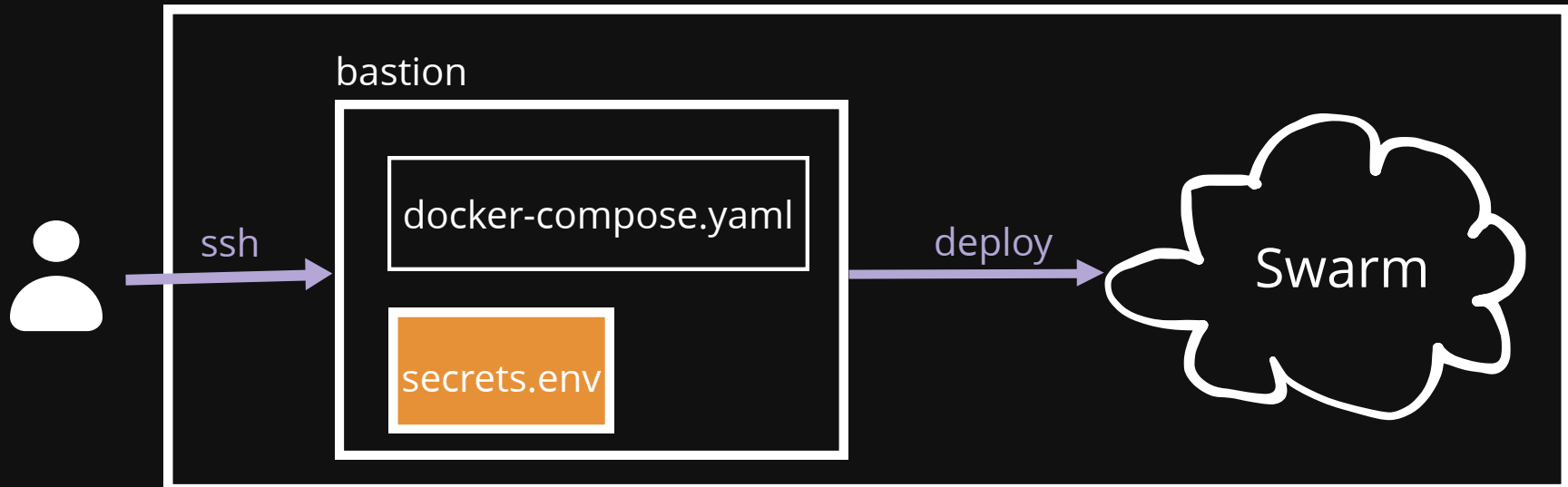
Because we now have many clients, we have to handle more than 300 secrets.

Back in the old days

Staging (on 40 dev laptops)



Prod



Speaker notes

So how do we handle those secrets in our stack?

In two thousands sixteen, we have setup docker and docker-compose to run our stack.

In a classic way, we have one git repo per microservice with one Dockerfile.

We used docker-compose to build and push docker images to our registry but we also used it to run the application locally.

We stored our secrets in plain text environment files on our laptop.

<***> We also used docker-compose and environment files on production.

To deploy a new release, you had to connect with ssh to a bastion, the security was done with an ssh configuration.

Then you update the docker-compose yaml file with new image versions.

And then update the secrets manually before running a docker-compose run.

Then it was deployed on an on-premise swarm cloud.

This have worked well for us up to last year when the dev team had grown quickly.

We exceeded 100 microservices for a standart installation with many databases such as elasticsearch.

<***> And unfortunately, what should happen, happened, dev laptops had not enough resources to run the application locally.

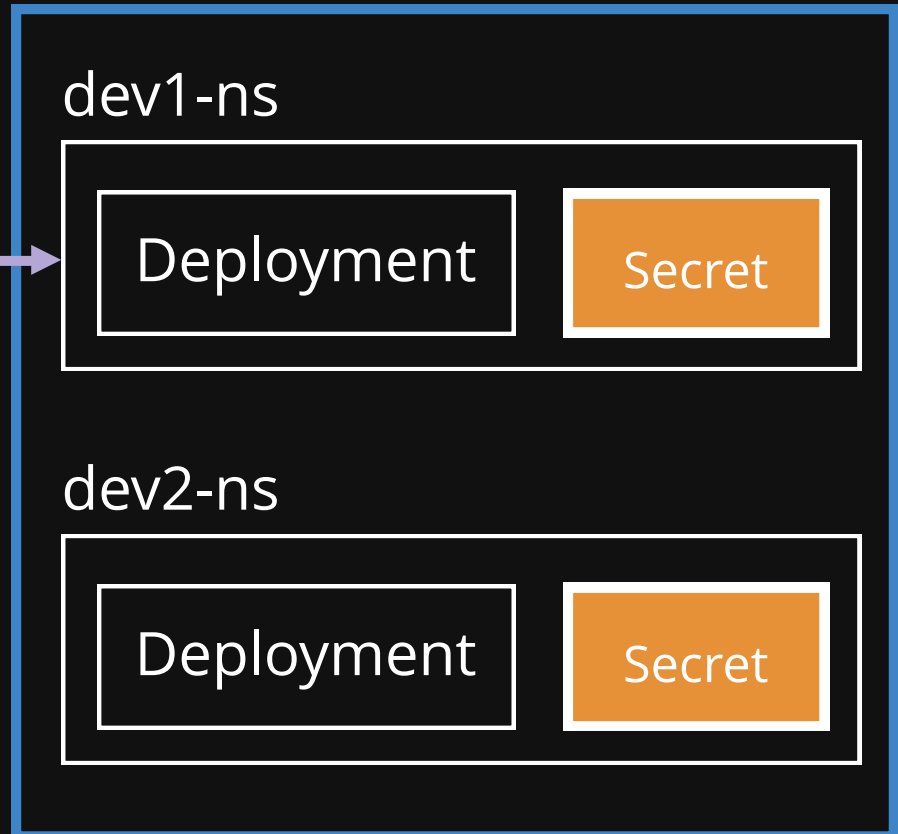
A new dev environment



Dev1 laptop



(2) skaffold
run



Speaker notes

It is one of the reasons why we switched to a managed kubernetes cloud.

Historically, our servers were hosted by OVH so we have chosen the OVH managed cloud.

Each developer has its own namespace and he can deploy his application inside.

We use skaffold and helm instead of docker-compose to build and deploy.

The new cluster is autoscalable, up and down, based on the Pods resources.

This is a really nice feature because we only pay for the resources we use.

How does it works in practice ?

On the morning, the cluster progressively grows as developers start working and launch their Pods.

At the end of the day, a CronJob shuts down all applications and the cluster fastly downscale to 1 machine.

With this solution, we have solved the resource limitation problem, but we also have a nice benefit: it's easier for people to collaborate

For example, if a colleague ask for help about the new feature I just pushed, I can inspect his running Pods with k9s, just as I do for my Pods.

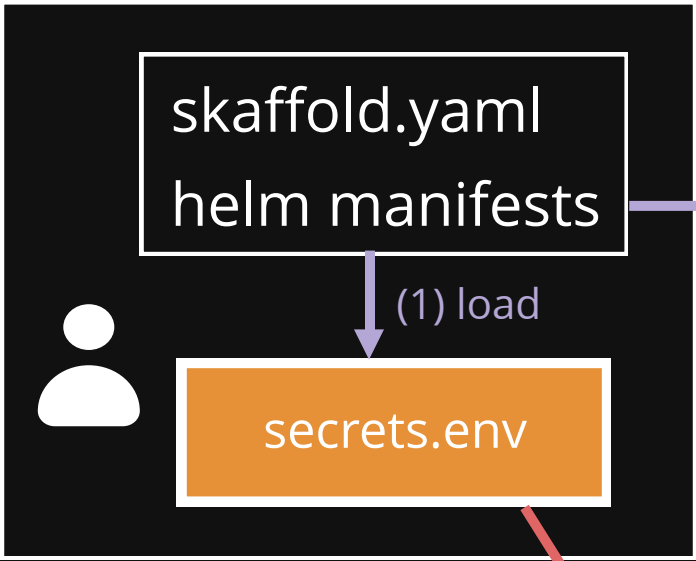
It's the same for data, I can connect to his mongodb instance just as I do with my mongodb instance.

So it's easier to help each other. At OneStock, it has created a positive dynamic inside the dev teams.

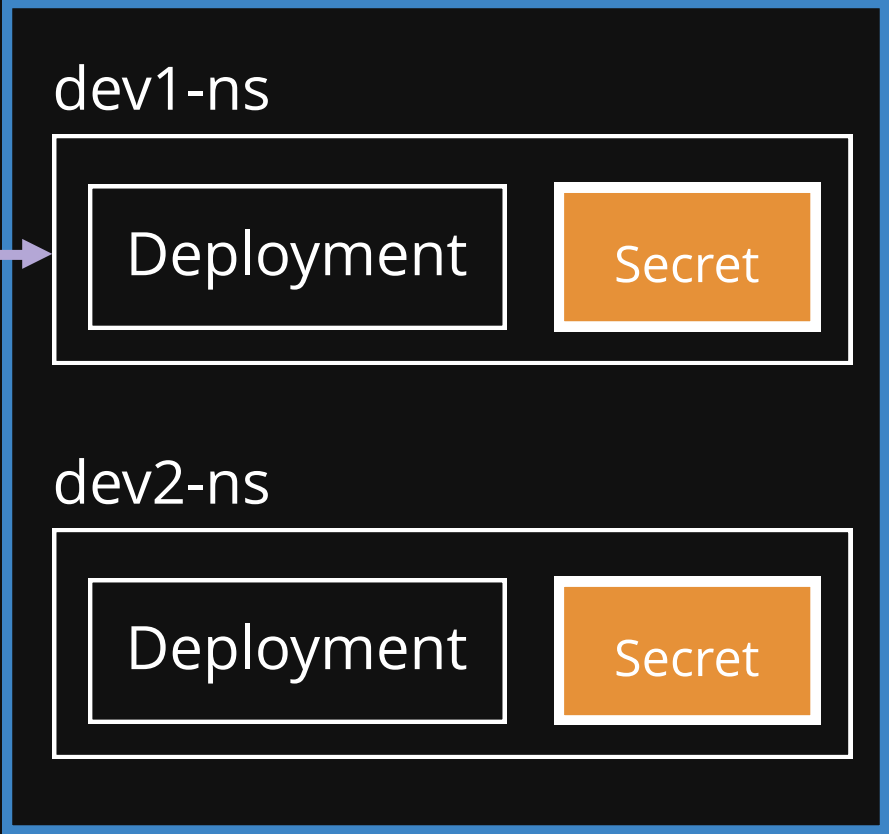
A password leak



Dev laptop



(2) skaffold
run



leak!



Speaker notes

But one day, my CTO came to my desk and told me:

Hey, someone from an untrusted IP is sending mail with our MailJet account !

After few investigations, the login/password have been leaked and a bad actor was using it !

Hopefully it was the dev account and only developers used it.

So it easy to fix the problem right ?

You change your credentials and you send a message to the dev channel.

People will update their local env files and everything is fine.

Right?

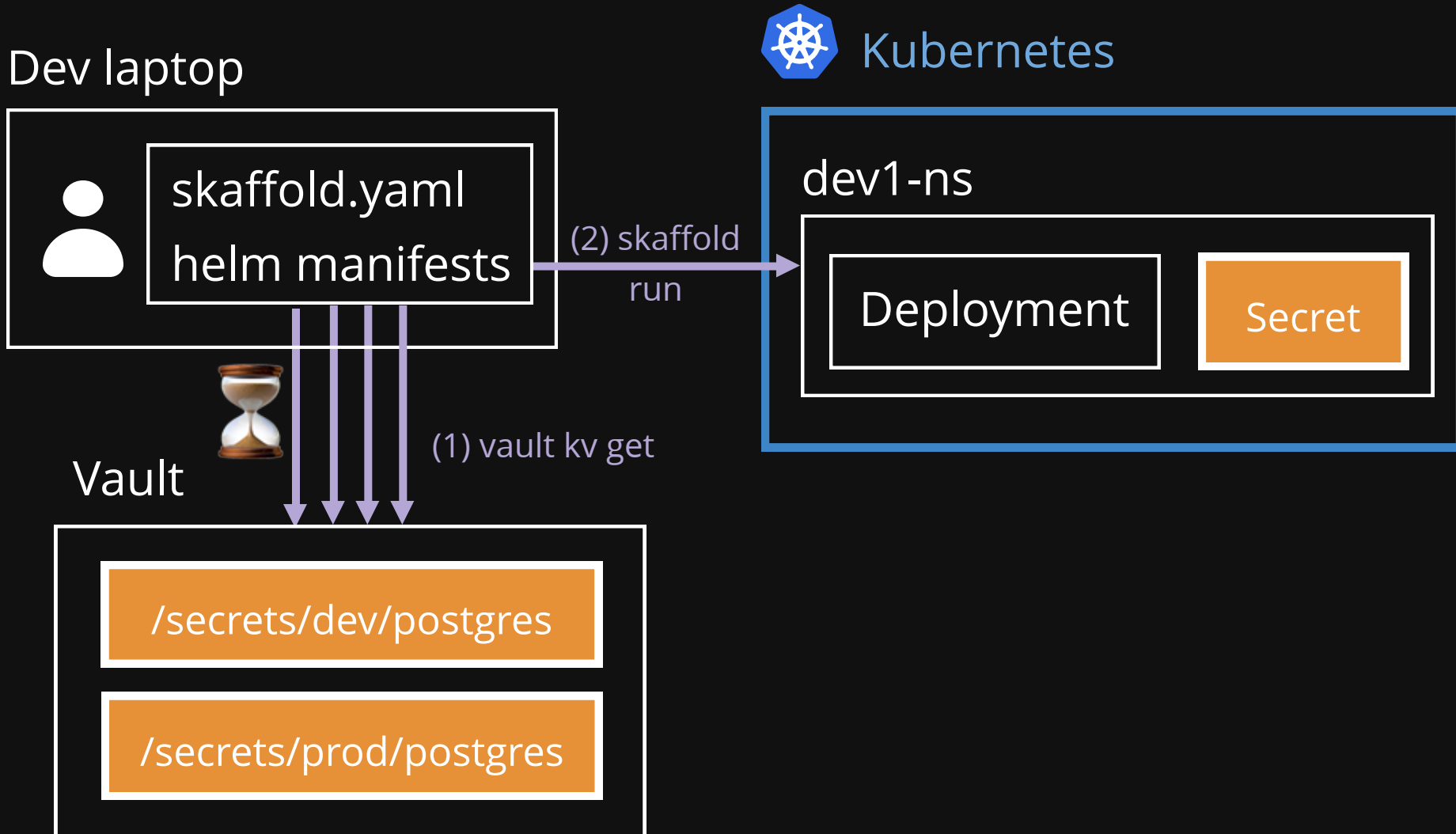
Well in practice, it was a pain.

People don't read all chat messages, they come back from holidays etc

I received messages during many weeks saying that mailing was broken.

So here, we had a secret rotation problem.

Vault



Speaker notes

That's why we use Vault.

For those who don't know, Vault is a secret management tool.

We have stored all our secrets there and we use the Vault policies to restrict secret access.

For example, only a few developers are allowed to access production secrets.

And as we already used LDAP for handling user identity, we just had to connect Vault to our existing LDAP.

So all devs can authenticate to Vault with their existing password, this is very convenient.

We just have to add a skaffold pre-install hook to fetch the credentials and we are done !

It was great for us because it has removed frictions and manual secret management for developers.

Now, we can update a secret in Vault and all new deployment will use the new value without any manual intervention.

<***> But we have a problem with this solution.

Every time a developer deploys an application, the hook makes many calls to Vault and it increased the deployment time by many seconds.

External secrets



(1) skaffold run

ExternalSecret
"postgres"

(3)
create

Secret
"postgres"

Vault

/secrets/dev/postgres

/secrets/prod/postgres

(2) fetch

➔ Developer commands

➔ External secrets operator

Speaker notes

To improve this we have setup the External Secret Operator.

The goal of this kubernetes operator is to synchronize secrets from external APIs into Kubernetes. So in our case, we will use the operator to synchronise a vault key to a kubernetes Secret object.

<***> So instead of directly declaring the Secret object, helm declares an ExternalSecret object with a vault path. When the developer deploys an application with skaffold run, it creates an ExternalSecret object.

<***> The external secret operator will fetch the value in vault and create the corresponding Secret object in kubernetes.

The pods definitions are not modified and they use the secret as they did before.

So great, we moved a synchronous action that slowed the application deployment to an asynchronous action made by a controller in kubernetes.

I'm happy because I know it's an important factor of adoption.

The faster the deployment is, the more developers will use the platform.

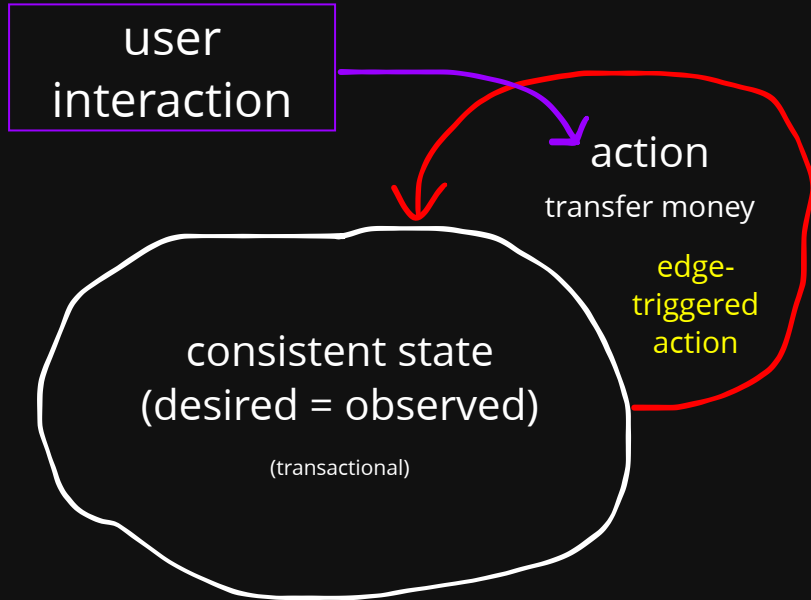
On top of being asynchronous, the operator is self-healing. For example, if a new password leak happens and that you decide to rotate the password in Vault, there will be an inconsistency between the password in Vault and the secrets in Kubernetes, but external-secrets operator will self-heal. Would you like to know more about that?

Yes!

Self-healing, Consistency, Desired vs. Observed state

Bank

Desired state **=** Observed state
"sum of balances is 0" "sum of balances in DB"
constraint `SUM(balance)` **fact**
`FROM accounts;`

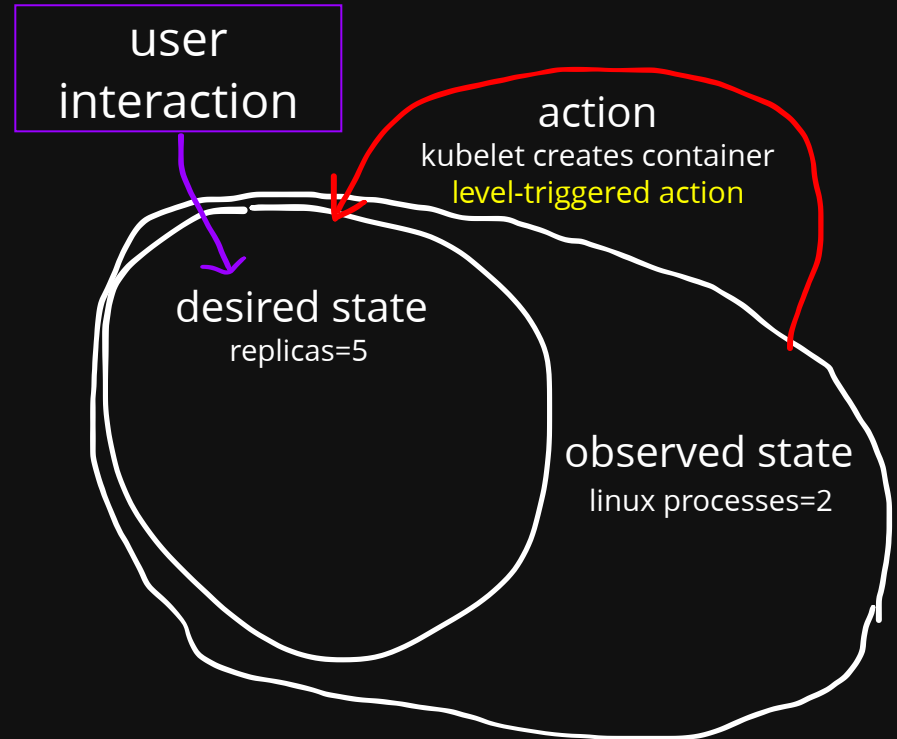


not able to recover data inconsistencies

but always consistent

Kubernetes

Desired state **≠** Observed state
"replicas=5" "linux processes=2"



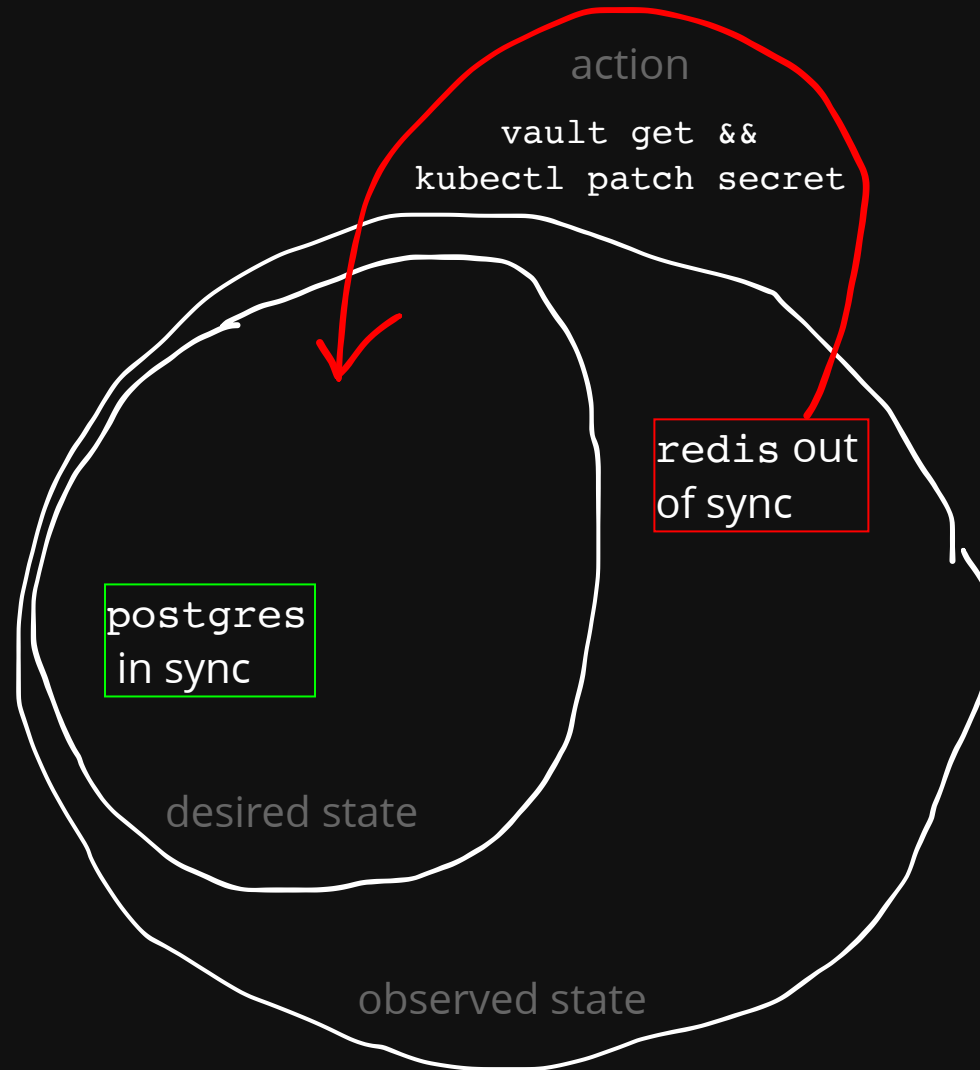
no data consistency

but can recover from inconsistencies

Speaker notes

Mael: for example, take a bank.

Self-healing in external-secrets operator



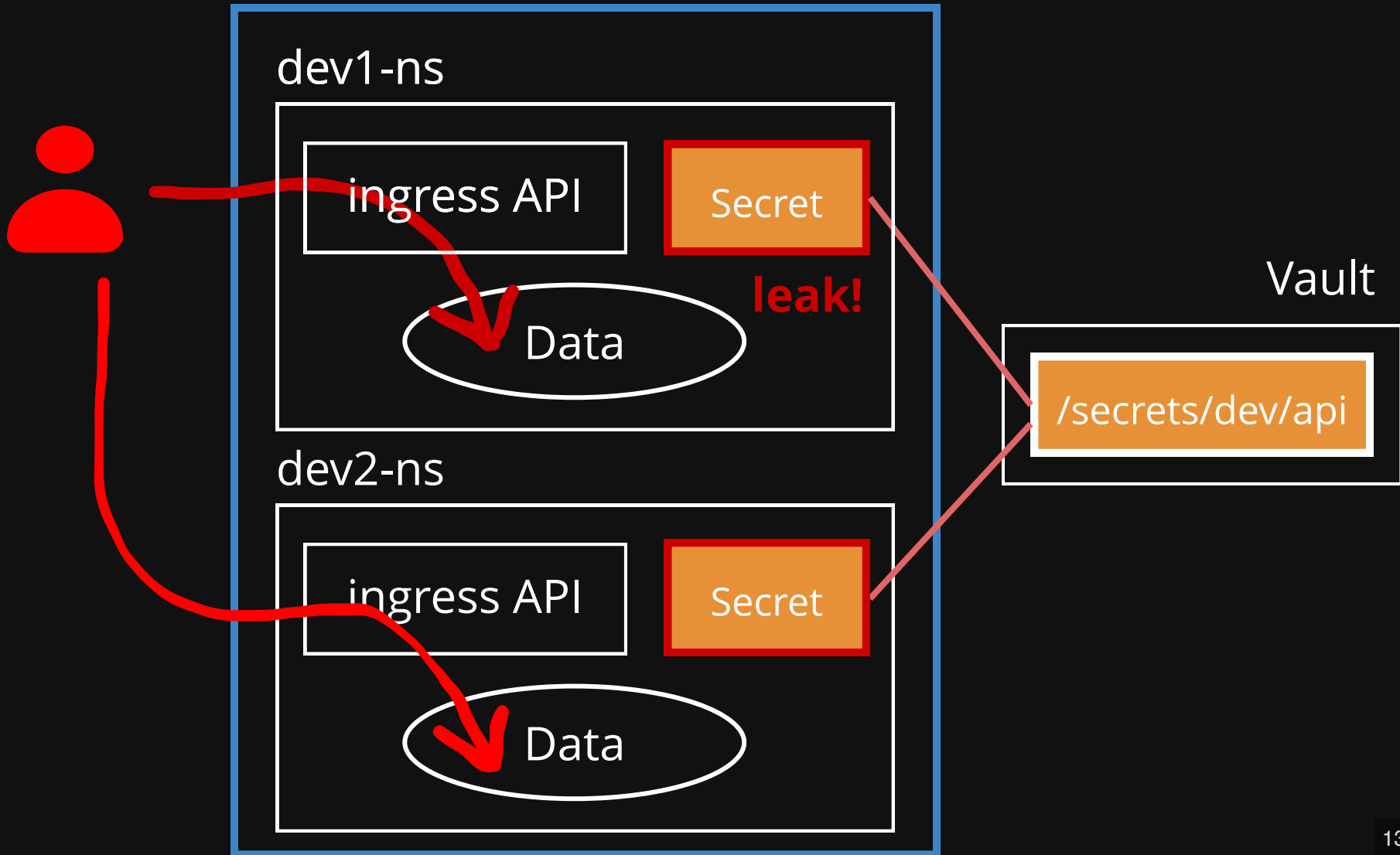
`postgres in sync` password in Vault matches Secret in Kubernetes

`postgres out of sync` password in Vault does not match Secret in Kubernetes

Speaker notes

Mael: n/a

A bad actor



Speaker notes

Thanks Mael for those precisions.

The 40 developers have used the External Secret Operator for last months and they are happy with it.

After the password leak I mentioned earlier, we decided to improve the security in the company.

We thought about the following scenario:

If someone with bad intentions manage to acces the kubernetes cluster and manage to compromise the API credentials of a developer, he could access to this developer dev data by making api calls.

<***> But as all developers use the API credentials defined in Vault, the bad actor could also access to all developers data.

This does not looks good.

And OneStock has many different teams and each teams handle different kind of data.

Some may be sensible even in a dev environment.

This security issue had been highly prioritized.

Ok, so we have to fix it.

Secrets at OneStock (bis)

Internal secrets

- OneStock API
- Postgre
- MongoDB
- ElasticSearch
- ...

10 secrets

40 developers

=> 400 random passwords

External secrets:

- Stock API
- SFTP
- Mailjet
- Gmaps
- ...

5 secrets

60 clients

Speaker notes

Well we cannot touch the external secrets as it refers to external services.

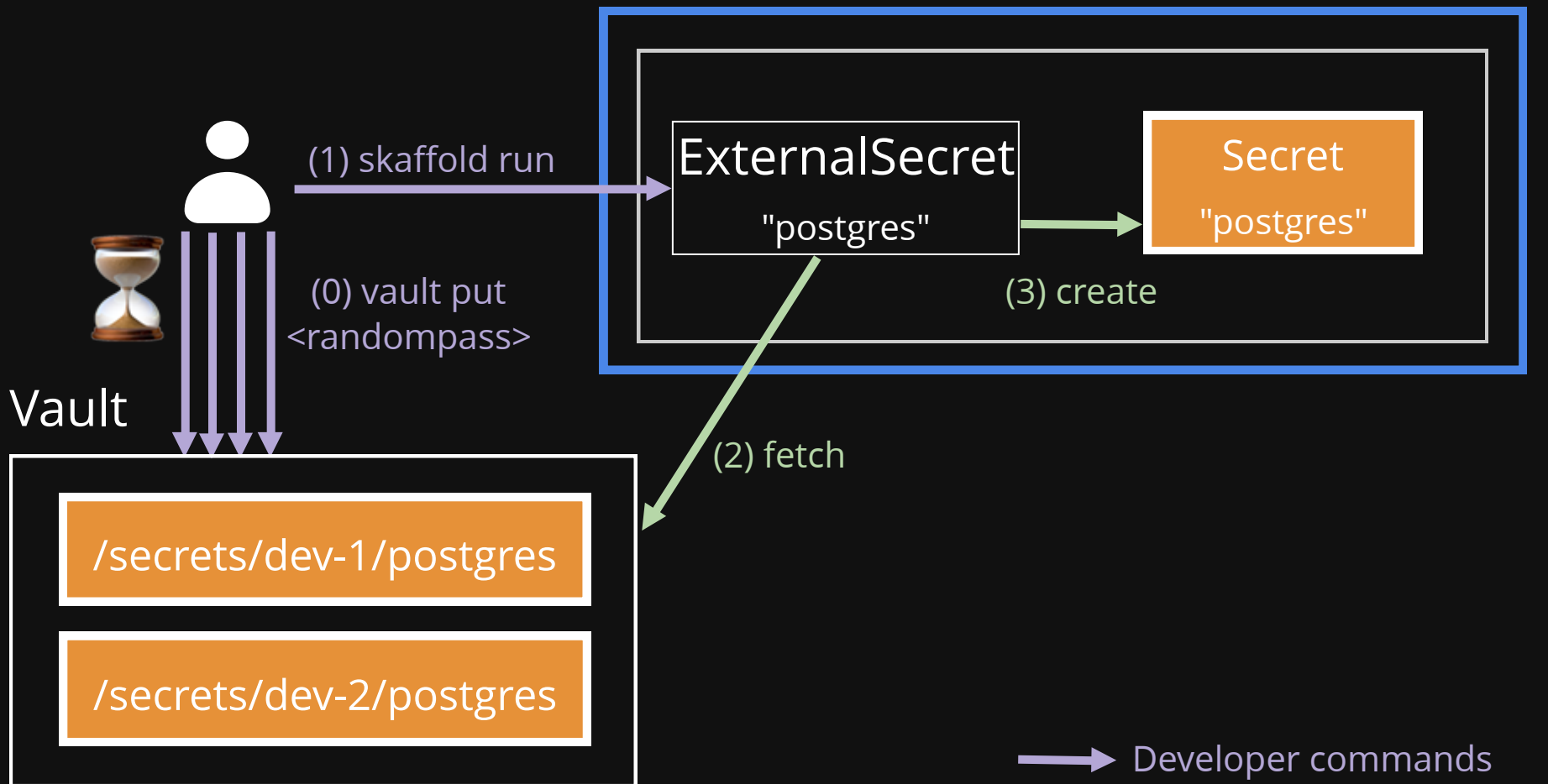
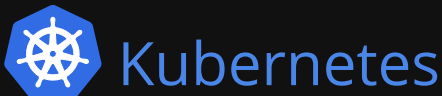
So we speak about internal secrets here.

We decided to generate random secrets for every developers.

So 10 secrets cross 40 developers, we have now 400 passwords to generate.

Ok, so how can we handle those new secrets ?

Slow solution



Speaker notes

Let's go back to the previous diagram.

Now, we have to add the username in the secret paths in Vault because each developer will have its own randomly generated passwords.

We cannot generate them once for all because new developers arrive and some other leave the company.

And even at a developer level, we cannot generate those passwords once for all.

What happens if I pull the product repo and someone has defined a new secret ?

I will deploy the new External Secret object in kubernetes

but the ExternalSecretOperator will fail to fetch the vault secret

because it doesn't exist in my username path.

<***>

So I have no choice, at every deployment, a pre-install hook must check the existence of the secrets in vault and generate them if it doesn't exist.

But it's the same problem than we saw earlier when we did many vault get during the deployment

Can we make this action in a controller ? It would be asynchronous and selfhealing.

I have searched on the web but I haven't found a controller that does this.

Mael:

On a rien trouvé sur internet de controller qui permette d'initialiser un ES avec un password aléatoire

Antoine:

Mael mais tu délirés, un controller, j'ai jamais contribué au projet kubernetes, j'ai fait que l'utiliser

Defining a controller: what are the desired and observed states?

Desired state: "No external secret is stuck with 'secret not found' due to a missing secret in Vault."

```
1 apiVersion: external-secrets.io/v1beta1
2 kind: ExternalSecret
3 status:
4   conditions:
5     - type: Ready
6       status: False
7       reason: SecretSyncedError
8       message: Secret key was not found
```

This means 'secret not found'

```
$ kubectl get externalsecret
```

NAME	KEY	PROPERTY	READY	REASON	MESSAGE
redis	secret/dev-1/redis	password	True	SecretSynced	Secret was synced
postgres	secret/dev-1/postgres	password	False	SecretSyncedError	Could not get secret data

Observed state: "Run `kubectl get externalsecret` and I look for `SecretSyncedError`"

Action: "Run `vault put password=random`"

```
$ vault kv put secret/dev-1/postgres password=random
==== Metadata =====
Key                    Value
---                    -
created_time          2022-06-26T15:37:26.01313574Z
custom_metadata       <nil>
```

Speaker notes

TL;DR: we don't "vault put" on the critical path, we wait until we get the error "secret not found" before doing something.

1) Show what desired state means: "There is no SecretSyncedError in ExternalSecret"

2) How to build the observed state: `kubectl get es` and check whether the ready condition has the reason "SecretSyncedError".

→ show `kubectl get l jq .conditions select SecretSyncedError`

3) Reconciliation actions = `vault put`

kubectl --watch to avoid polling ExternalSecrets

Get alerted as soon as `SecretSyncedError` appears

<https://asciinema.org/a/504357/iframe?autoplay=1>

Writing our one-liner controller

Observe state

because we are piping jq

so that we can use jq

```
1 kubectl get externalsecret --watch -ojson \
2 | jq 'select(.status.conditions[]?.reason == "SecretSyncedError")' --unbuffered \
3 | jq '.spec.data[0].remoteRef' --unbuffered \
4 | jq '"\(.key) \(.property)"' -r \
5 | while read key property
6 do
7   vault kv put $key $property=somerandomvalue
8 done
```

Action

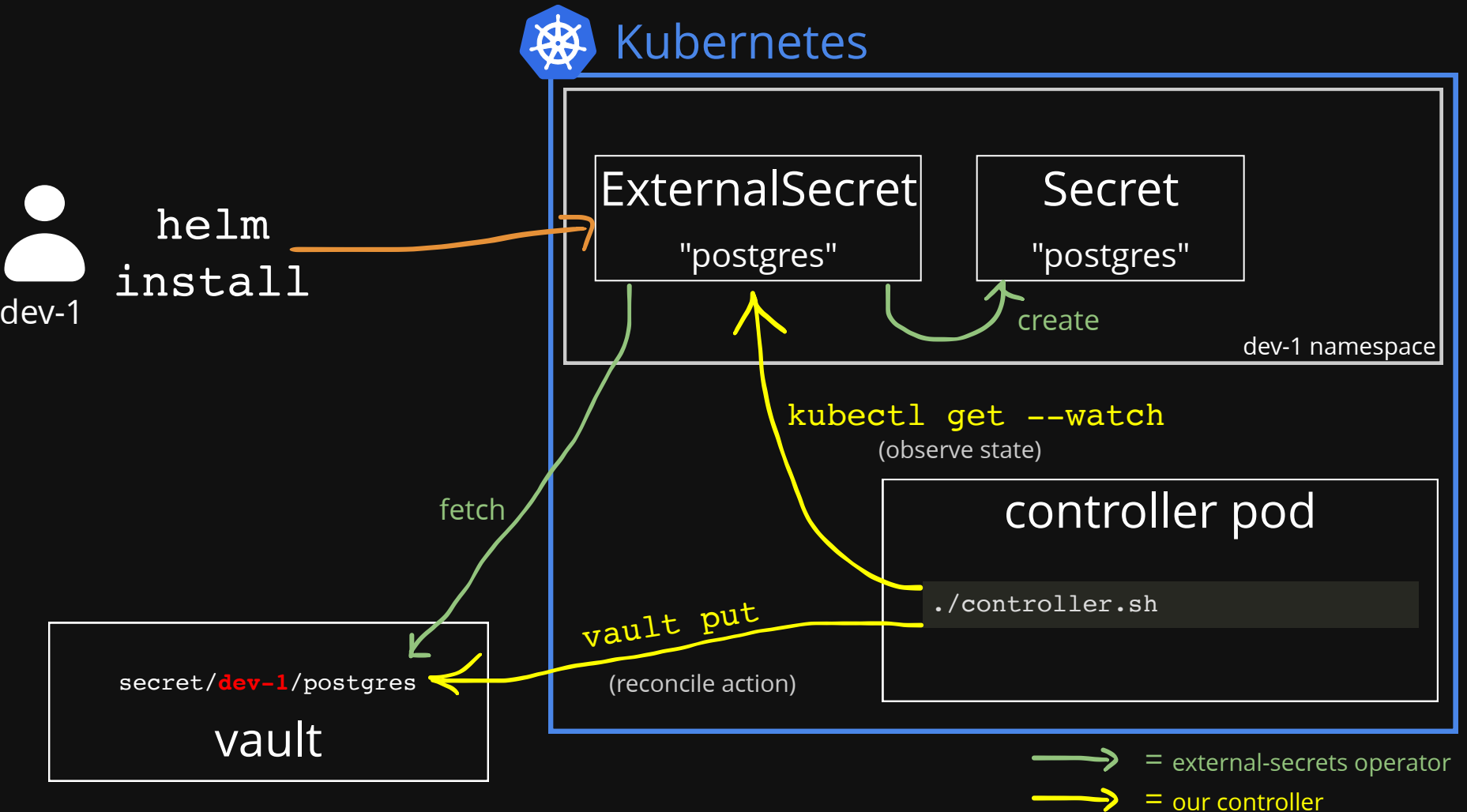
We only need to take action when **SecretSyncedError** exists

Our one-liner controller in action!

<https://asciinema.org/a/504076/iframe?autoplay=1&t=25s>

A real controller runs inside a Pod, right?

Visualising the controller pod in the cluster



A real controller runs inside a Pod, right?

Let us write a **Dockerfile** and a **Deployment** manifest

```
#!/bin/bash

kubect1 get externalsecret --watch -ojson \
| jq 'select(.status.conditions[ ]?.reason == "SecretSyncedError")' --unbuffered \
| jq '.spec.data[0].remoteRef' --unbuffered \
| jq '"\(.key) \(.property)"' -r \
| while read key property
do
  vault kv put $key $property=somerandomvalue
done
```

controller.sh

```
FROM alpine:3.16
```

Dockerfile

```
# The "setcap -r" is detailed in https://github.com/hashicorp/vault/issues/10924.
RUN tee -a /etc/apk/repositories <<<"@testing http://dl-cdn.alpinelinux.org/alpine/edge/testing"
  && apk add --update --no-cache bash curl jq kubect1@testing vault libcap \
  && setcap -r /usr/sbin/vault

COPY controller.sh /usr/local/bin/controller.sh
CMD ["controller.sh"]
```

```
apiVersion: apps/v1      deploy.yaml
kind: Deployment
metadata:
  name: controller
spec:
  replicas: 1
  selector:
    matchLabels: {name: controller}
  template:
    metadata:
      labels: {name: controller}
    spec:
      containers:
        - name: controller
          image: controller:local
          imagePullPolicy: Never
          env:
            - name: VAULT_ADDR
              value: http://vault.vault:8200
            - name: VAULT_TOKEN
              valueFrom:
                secretKeyRef:
                  name: vault-token
                  key: vault-token
              serviceAccountName: controller
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: controller
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: controller
subjects:
  - kind: ServiceAccount
    name: controller
roleRef:
  name: external-secrets-reader
  kind: Role
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: external-secrets-reader
rules:
  - apiGroups: [external-secrets.io]
    resources: [externalsecrets]
    verbs: [get, list, watch, update, patch]
```

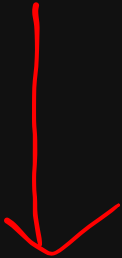
A real controller runs inside a Pod, right?

The controller pod in action

<https://asciinema.org/a/504467/iframe?autoplay=1>

What now?

Users won't know
when something
goes wrong



Use conditions to
alert user when
something goes
wrong

```
1 apiVersion: external-secrets.io/v1beta
2 kind: ExternalSecret
3 metadata:
4   name: postgres
5 spec:
6   data:
7     - remoteRef:
8       conversionStrategy: Default
9       key: secret/dev-1/postgres
10      property: password
11      secretKey: password
12    refreshInterval: 5s
13    secretStoreRef:
14      name: vault-backend
15    target:
16      name: postgres
17  status:
18    conditions:
19      - type: Ready
20        status: False
21        reason: SecretSyncedError
22        message: Secret key was not found
23      - type: Created
24        status: False
25        reason: VaultConnError
26        message: Vault returned 403 unauthorized
27
```

See [controller-with-conditions.sh](#)

What now?

Slow
sequential
processing

due to the while loop




Use Go and
controller-runtime

no more slow sequential processing, i.e., controller can handle hundreds of ExternalSecrets

```
1 func main() {
2     mgr, _ := manager.New(config.GetConfigOrDie(), manager.Options{
3     c, err := controller.New("ext-secrets-vault-creator", mgr, contr
4     Reconciler: reconcile.Func(func(ctx context.Context, r reconcil
5         extsecret := v1.ExternalSecret{}
6         err := mgr.GetClient().Get(ctx, r.NamespacedName, &secret)
7
8         // vault kv put
9
10        return reconcile.Result{}, nil
11    })),
12 })
13 }
```




Antoine Le Squéren

 [in/antoine-le-squeren-8495a774](https://www.linkedin.com/in/antoine-le-squeren-8495a774)

Maël Valais

 [@maelvl](https://twitter.com/maelvl)  [in/maelvalais](https://www.linkedin.com/in/maelvalais)

Speaker notes

Thanks Mael for this explanation.

I have tested the onliner bash controller inside our cluster and it worked well.

But in order to use it for real, I mean with all developers and for a long run, I will definitively implement the tiny go controller.

It would be easier to test and maintain, especially at OneStock because most of developers write go code every day so the knowledge and tools are already here.

By the way, if you are looking for a job in golang development or devops, OneStock has open positions so feel free to contact me or the company. Fullremote is accepted.

I'm really happy to have met Mael.

We still speak about software problems during coffee breaks and we still help each other.

It's nice to see people from outside of your company, it opens perspectives and it may give you new ideas.

And we think coworking spaces are a great place for that.

Thank you for your attention