Beesting

Can't touch this

Image by mariya_m on pixabay

Who am I?

Software engineer turned Cloud Enthusiast ☁️

Kubernetes wizard 🪄

Linux Nerd 🐧

Let me take you back to

August 2024

Context-aware security incident response with Dynatrace Automations and Tetragon

Published May 3, 2024 | Updated December 12, 2024 | 11 min read

Mario Kahlhofer Simon Ammer Markus Gierlinger

Application security Engineering

In this blog post

For the most severe threat scenarios, you want multiple layers of automated defenses, and not have to rely on humans to analyze the traces of an attack weeks after your system got compromised. Many security teams use runbooks to glue together tools, processes, events, and actions for security incident response. A runbook lays out the step-by-step instructions to follow when a security incident happens, when an emerging threat surfaces, or when your security tool reports suspicious behavior.

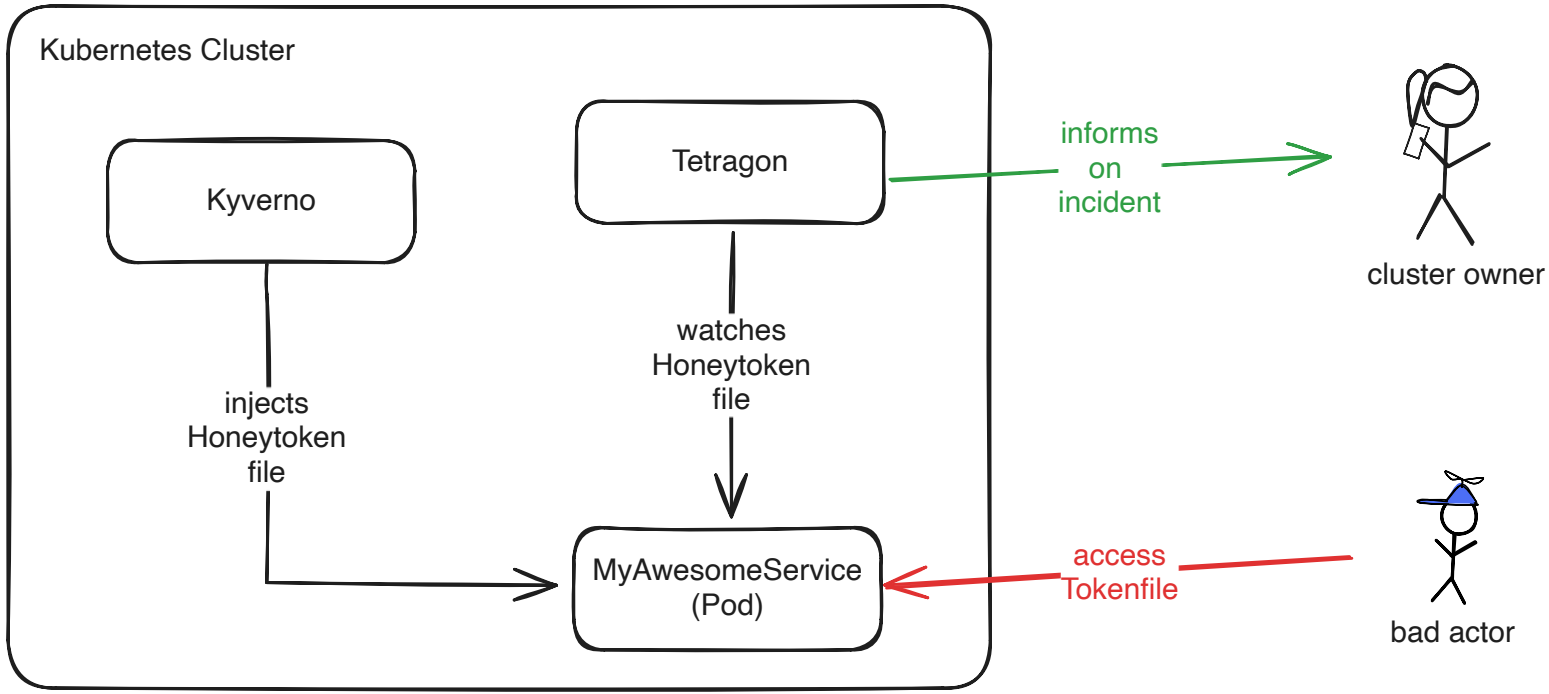
But runbooks that stitch together glamorous security tooling are merely decorations without automated workflows for incident detection and response.

The security community agrees on many high-level best practices in such situations, but we need a single platform solution to orchestrate application security, observability, and DevOps practices. Because every situation is a little unique, Dynatrace makes it easy to create custom runbooks using Dynatrace Automations, fine-tuned to your individual business risks.

In this blog post, we'll demonstrate how to use Dynatrace Automations to build a runbook that combats sophisticated security incidents with honeytokens and eBPF-based detection. We show an end-to-end solution, starting with deploying policies in a Kubernetes cluster and ending in a pull request assigned to the responsible team, all without manual intervention.

To demonstrate the integration of external security tools into the Dynatrace platform, we use Tetragon for eBPF-based security monitoring. Using Kyverno, we can automatically kick attackers out of our cluster with network policies and harden our configuration with a GitOps workflow to prevent the same incident from happening again.

Link to blog


Kubernetes Cluster
Kyverno
Tetragon
informs
on
incident
cluster owner
watches
Honeytoken
file
injects
Honeytoken
file
MyAwesomeService
(Pod)
access
Tokenfile
bad actor

What even is a Honeytoken?

Image by Freepik on Freepik

Digital bait

Image by freepik on Freepik

commerce:SEO

We are currently updating our CMS database. You will soon find further information about **commerce:SEO CMS** on this page. We will provide you an overview with all details about:

- latest changes and bug fixes
- updated or new plugins
- templates and themes
- developers news

webEdition

We are currently updating our CMS database. You will soon find further information about **webEdition CMS** on this page. We will provide you an overview with all details about:

- latest changes and bug fixes
- updated or new plugins
- templates and themes
- developers news

Cuppa CMS

We are currently updating our CMS database. You will soon find further information about **Cuppa CMS** on this page. We will provide you an overview with all details about:

- latest changes and bug fixes
- updated or new plugins
- templates and themes
- developers news

H.H.G. multistore

CMS Categories

Blog (28)
Clan CMS (4)
CMF (1)
CMS / Portals (343)
CRM (1)
ECM (1)
eCommerce (22)
Forum (15)
Groupware (7)
Image Galleries (8)
Lite / Simple (11)
LMS / LCMS (5)
MVC (2)
Social Dating (1)
WAF (1)
Wiki (4)

Popular Demos

WordPress Demo
Drupal Demo
MODx Demo
Typo3 Demo
Joomla Demo
concrete5 Demo

New Demos

GitHub Advisory Database / Unreviewed / CVE-2022-41544

GetSimple CMS v3.3.16 was discovered to contain a remote...

Critical severity | Unreviewed | Published on Oct 18, 2022 to the GitHub Advisory Database • Updated on May 24, 2023

Package	Affected versions	Patched versions
No package listed— Suggest a package	Unknown	Unknown

Description

GetSimple CMS v3.3.16 was discovered to contain a remote code execution (RCE) vulnerability via the edited_file parameter in admin/theme-edit.php.

References

- https://nvd.nist.gov/vuln/detail/CVE-2022-41544
- GetSimpleCMS/GetSimpleCMS#1352
- http://packetstormsecurity.com/files/172553/GetSimple-CMS-3.3.16-Shell-Upload.html

Published by the National Vulnerability Database on Oct 18, 2022

Published to the GitHub Advisory Database on Oct 18, 2022

Last updated on May 24, 2023

Severity

Critical 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

EPSS score

4.568% (92nd percentile)

Weaknesses

No CWEs

CVE ID

CVE-2022-41544

GHSA ID

GHSA-c599-8gm5-c2jh

Source code

No known source code

```
$ root@simpecms-...:/root#
```

```
$ root@simpecms-...:/root# env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=simpecms-6c4bfc97cd-wdnhv
PHP_VERSION=7.4.9
APACHE_CONFDIR=/etc/apache2
PHP_LDFLAGS=-Wl,-O1 -pie
PWD=/var/www/html
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
TERM=xterm
PHP_URL=https://www.php.net/distributions/php-7.4.9.tar.xz
...
```

```
$ root@simpecms-...:/root# env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=simpecms-6c4bfc97cd-wdnhv
PHP_VERSION=7.4.9
APACHE_CONFDIR=/etc/apache2
PHP_LDFLAGS=-Wl,-O1 -pie
PWD=/var/www/html
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
TERM=xterm
PHP_URL=https://www.php.net/distributions/php-7.4.9.tar.xz
...
```

```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
```

```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
total 20
drwxr-xr-x 4 root root 4096 Jan  5 09:57 .
drwxr-xr-x 1 root root 4096 Jan  5 09:57 ..
drwxr-xr-x 2 root root 4096 Jan  5 09:57 eks.amazonaws.com
drwxr-xr-x 3 root root 4096 Jan  5 09:57 kubernetes.io
```

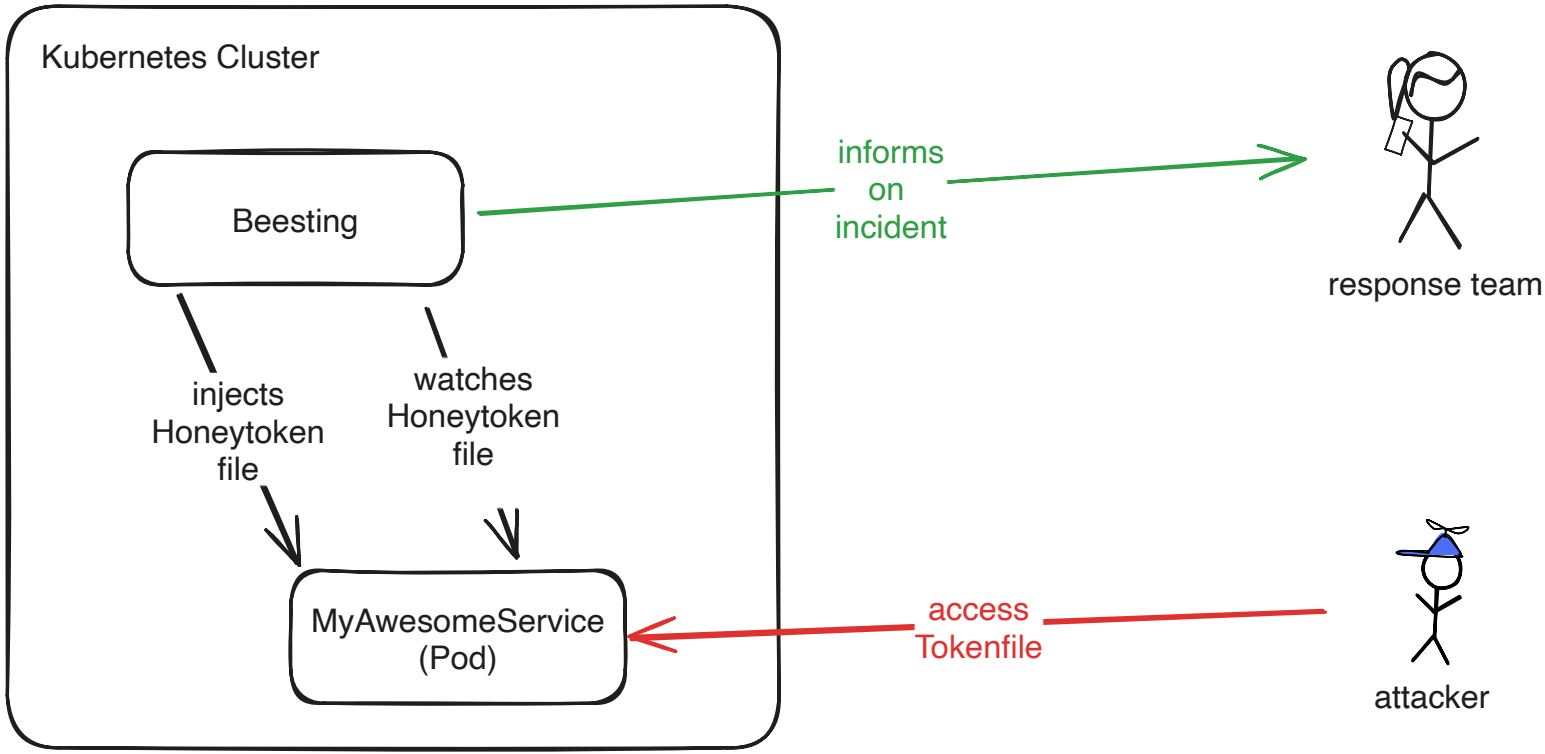
```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
total 20
drwxr-xr-x 4 root root 4096 Jan  5 09:57 .
drwxr-xr-x 1 root root 4096 Jan  5 09:57 ..
drwxr-xr-x 2 root root 4096 Jan  5 09:57 eks.amazonaws.com
drwxr-xr-x 3 root root 4096 Jan  5 09:57 kubernetes.io
```

```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
total 20
drwxr-xr-x 4 root root 4096 Jan  5 09:57 .
drwxr-xr-x 1 root root 4096 Jan  5 09:57 ..
drwxr-xr-x 2 root root 4096 Jan  5 09:57 eks.amazonaws.com
drwxr-xr-x 3 root root 4096 Jan  5 09:57 kubernetes.io

$ root@simpecms-...:/root# ls -al /var/run/secrets/eks.amazonaws.com/
total 12
drwxr-xr-x 2 root root 4096 Jan  5 09:57 .
drwxr-xr-x 4 root root 4096 Jan  5 09:57 ..
-rw-r--r-- 1 root root   16 Jan  5 09:57 access_key_token
```

```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
total 20
drwxr-xr-x 4 root root 4096 Jan  5 09:57 .
drwxr-xr-x 1 root root 4096 Jan  5 09:57 ..
drwxr-xr-x 2 root root 4096 Jan  5 09:57 eks.amazonaws.com
drwxr-xr-x 3 root root 4096 Jan  5 09:57 kubernetes.io

$ root@simpecms-...:/root# ls -al /var/run/secrets/eks.amazonaws.com/
total 12
drwxr-xr-x 2 root root 4096 Jan  5 09:57 .
drwxr-xr-x 4 root root 4096 Jan  5 09:57 ..
-rw-r--r-- 1 root root   16 Jan  5 09:57 access_key_token
```

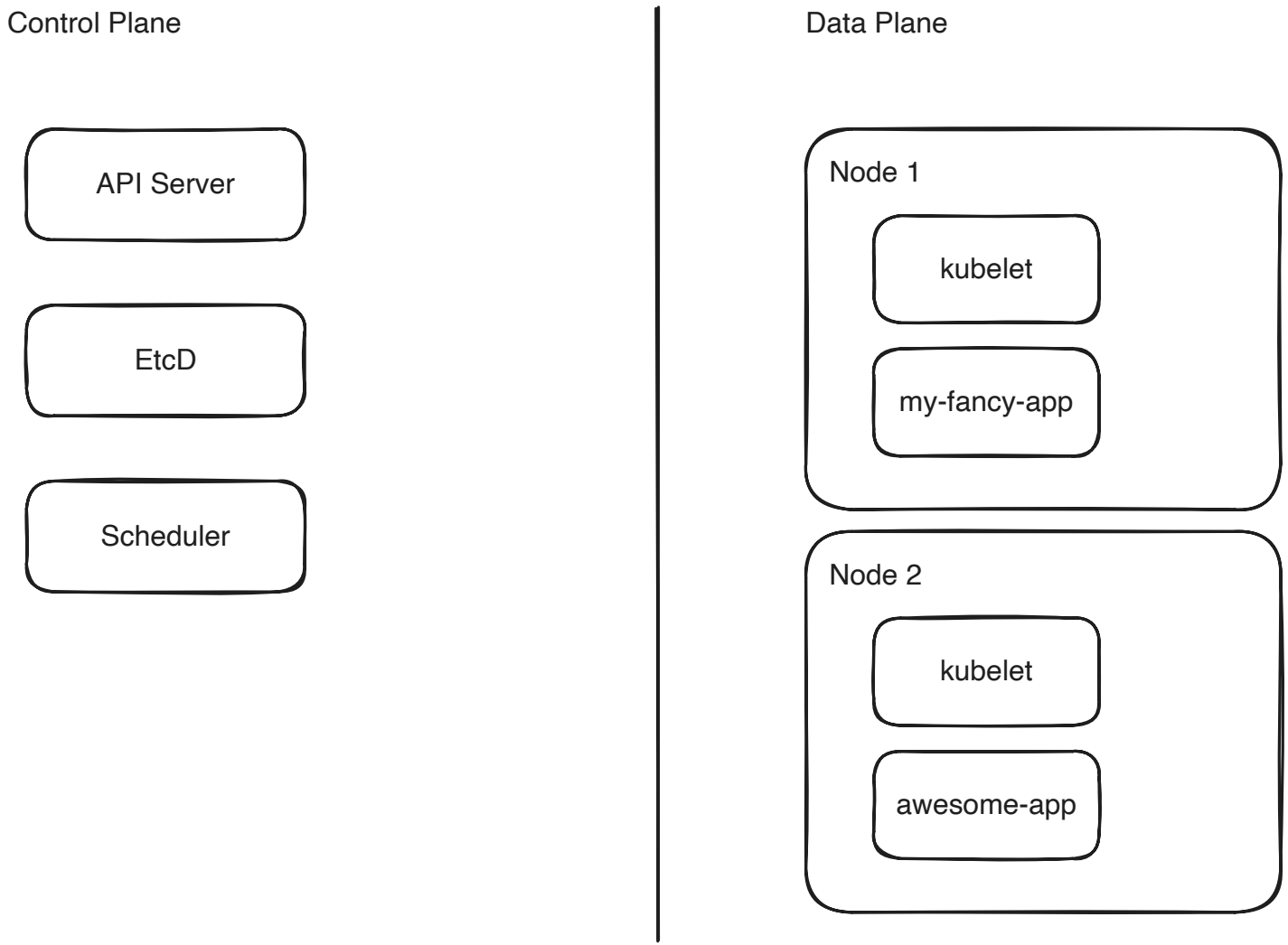
```
$ root@simpecms-...:/root# ls -al /var/run/secrets/
total 20
drwxr-xr-x 4 root root 4096 Jan  5 09:57 .
drwxr-xr-x 1 root root 4096 Jan  5 09:57 ..
drwxr-xr-x 2 root root 4096 Jan  5 09:57 eks.amazonaws.com
drwxr-xr-x 3 root root 4096 Jan  5 09:57 kubernetes.io

$ root@simpecms-...:/root# ls -al /var/run/secrets/eks.amazonaws.com/
total 12
drwxr-xr-x 2 root root 4096 Jan  5 09:57 .
drwxr-xr-x 4 root root 4096 Jan  5 09:57 ..
-rw-r--r-- 1 root root   16 Jan  5 09:57 access_key_token

$ root@simpecms-...:/root# cat /v/r/s/eks.amazonaws.com/access_key_token
```

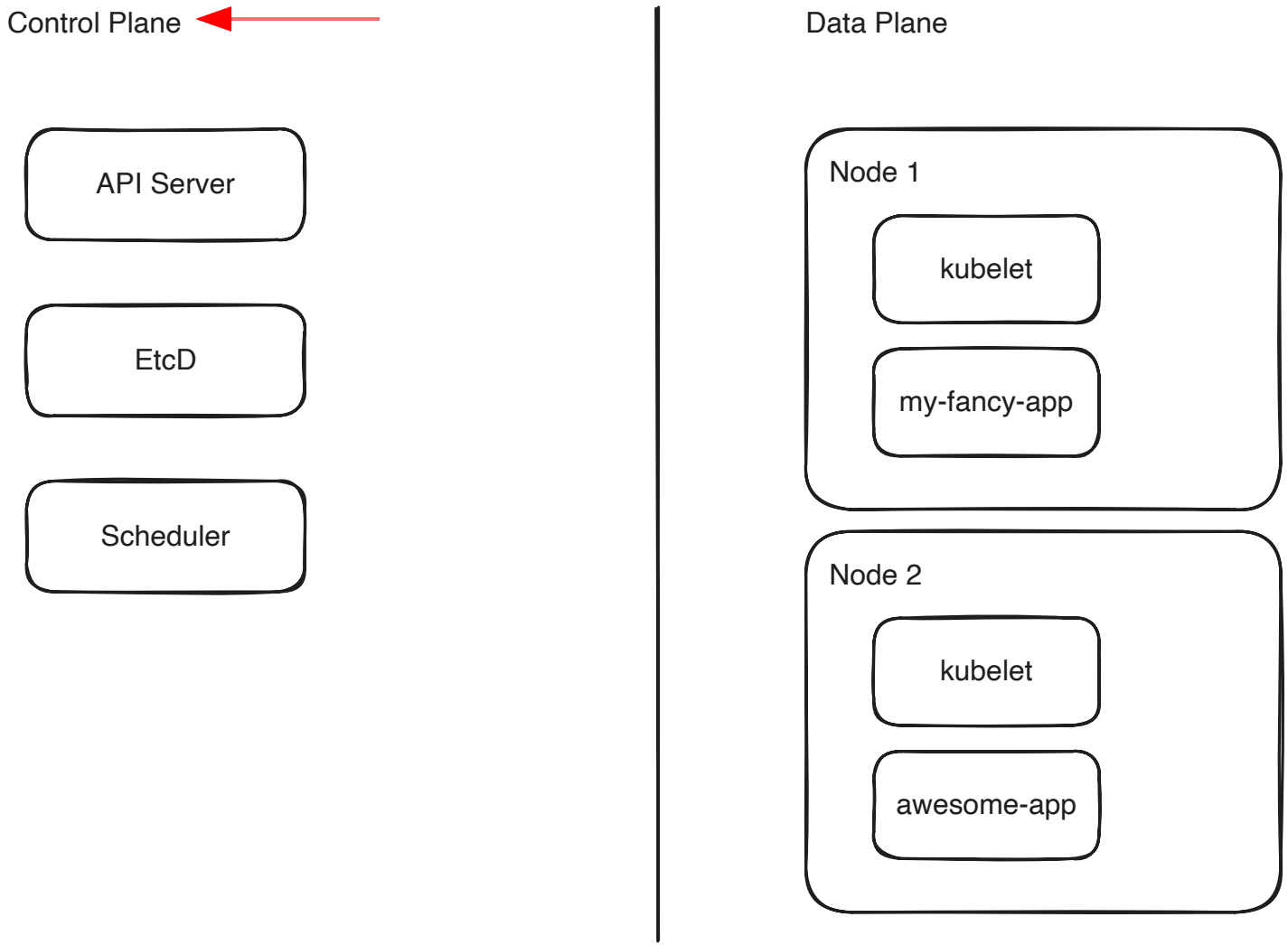

SECURITY TEAM
YOU JUST ACTIVATED
MY TRAP CARD

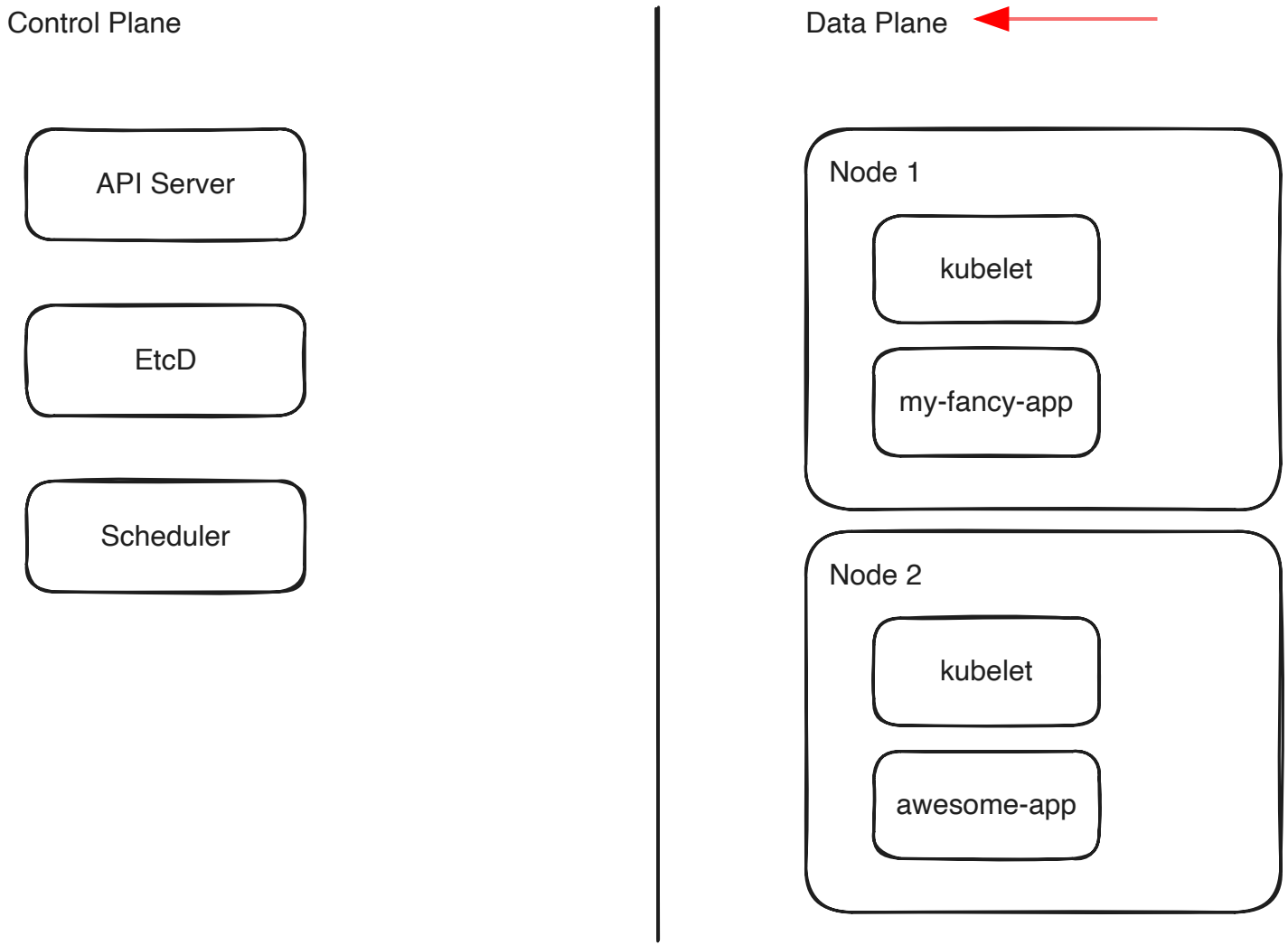

Kubernetes Cluster
Beesting
informs
on
incident
response team
injects
Honeytoken
file
watches
Honeytoken
file
MyAwesomeService
(Pod)
access
Tokenfile
attacker

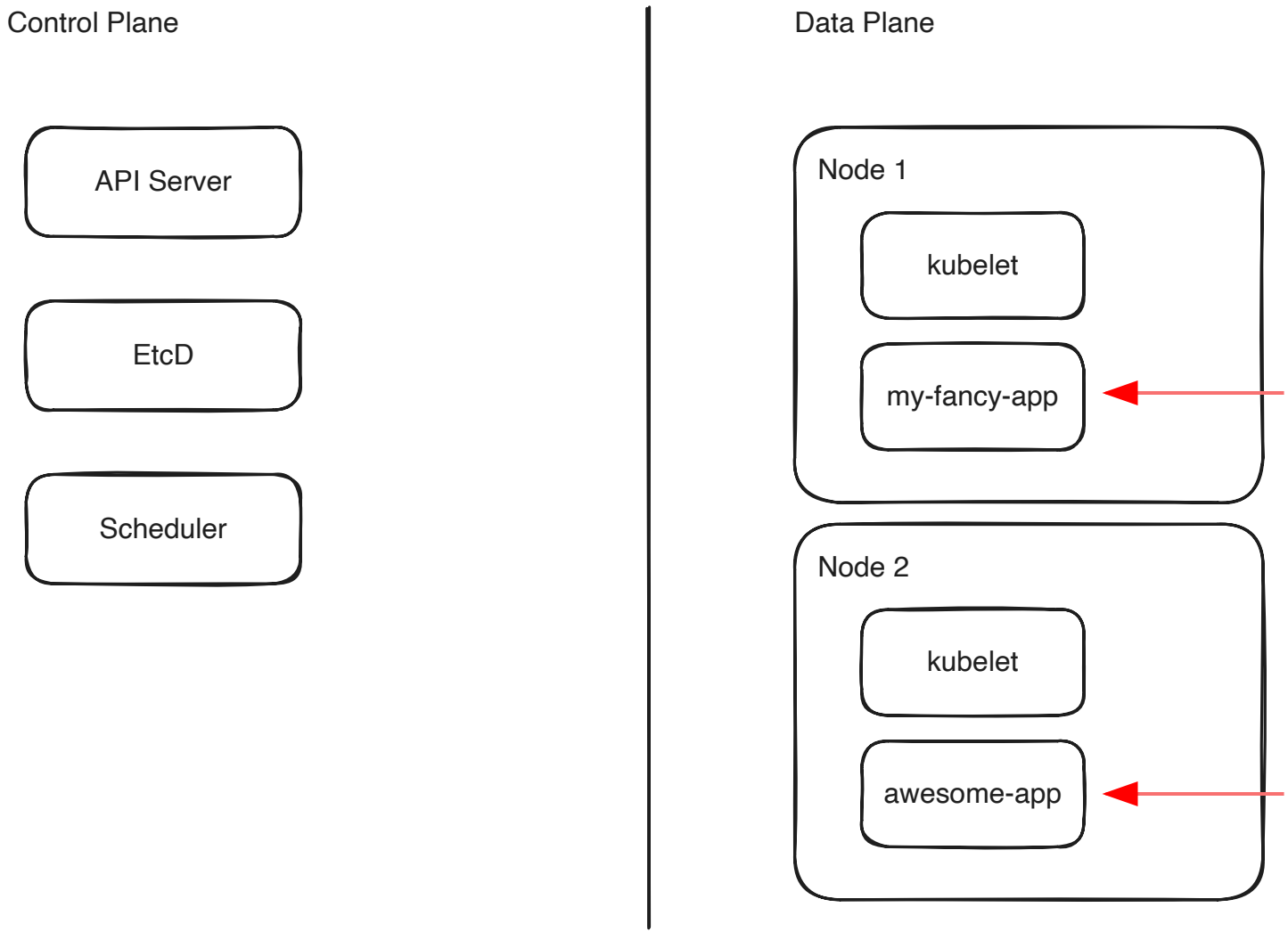
How does Kubernetes work?

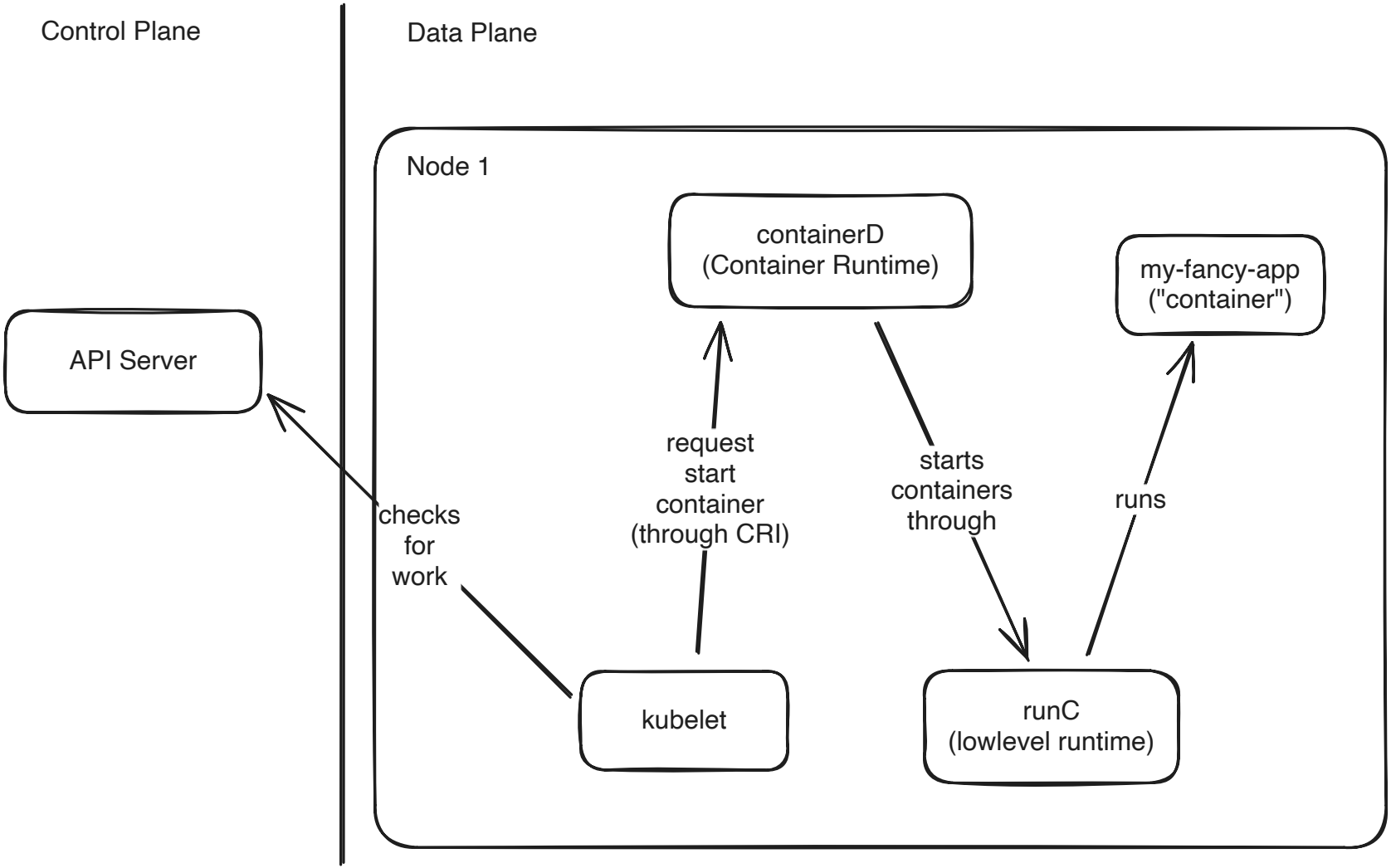
Image by tawatchai07 on Freepik

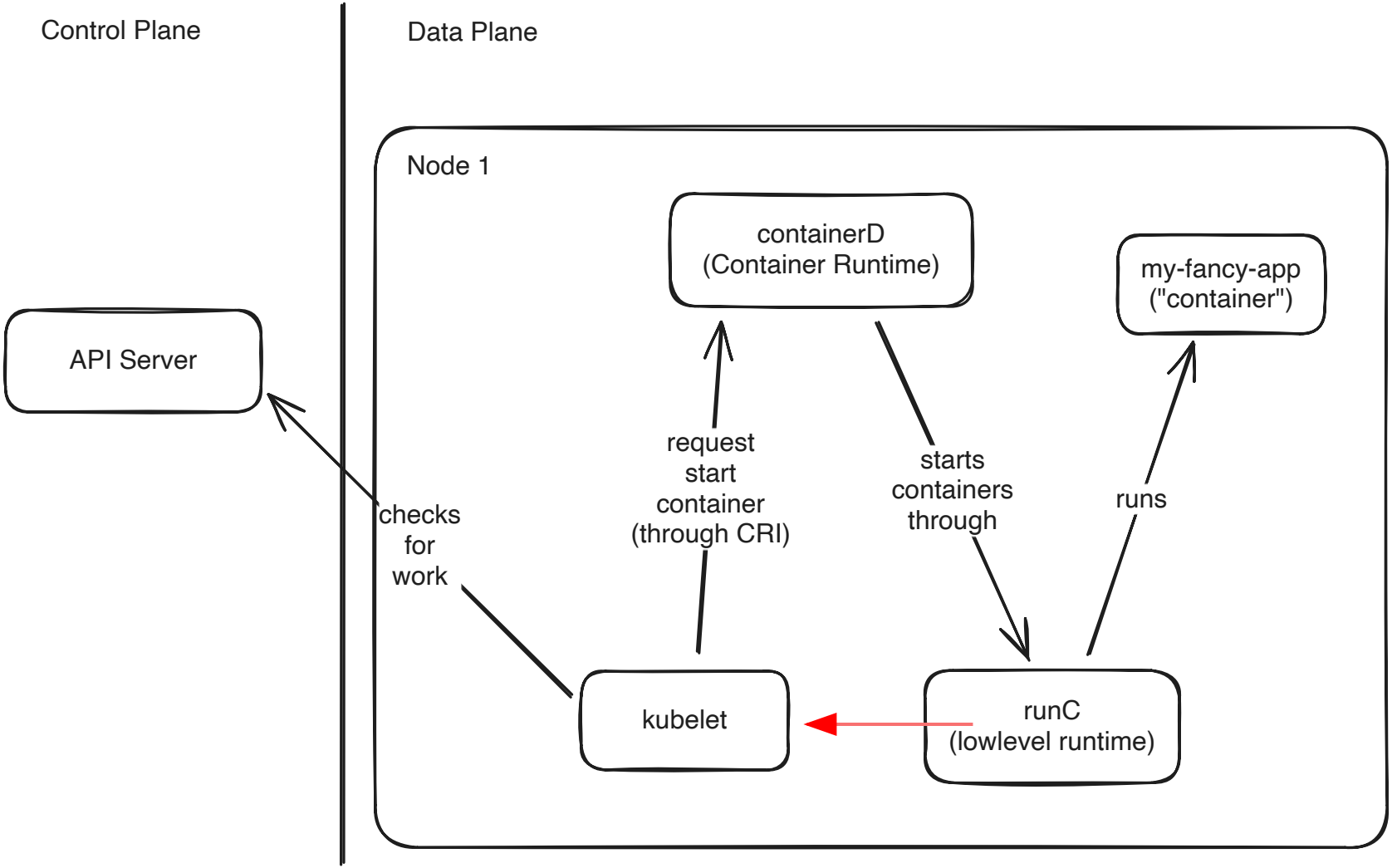

Control Plane
API Server
EtcD
Scheduler
Data Plane
Node 1
kubelet
my-fancy-app
Node 2
kubelet
awesome-app

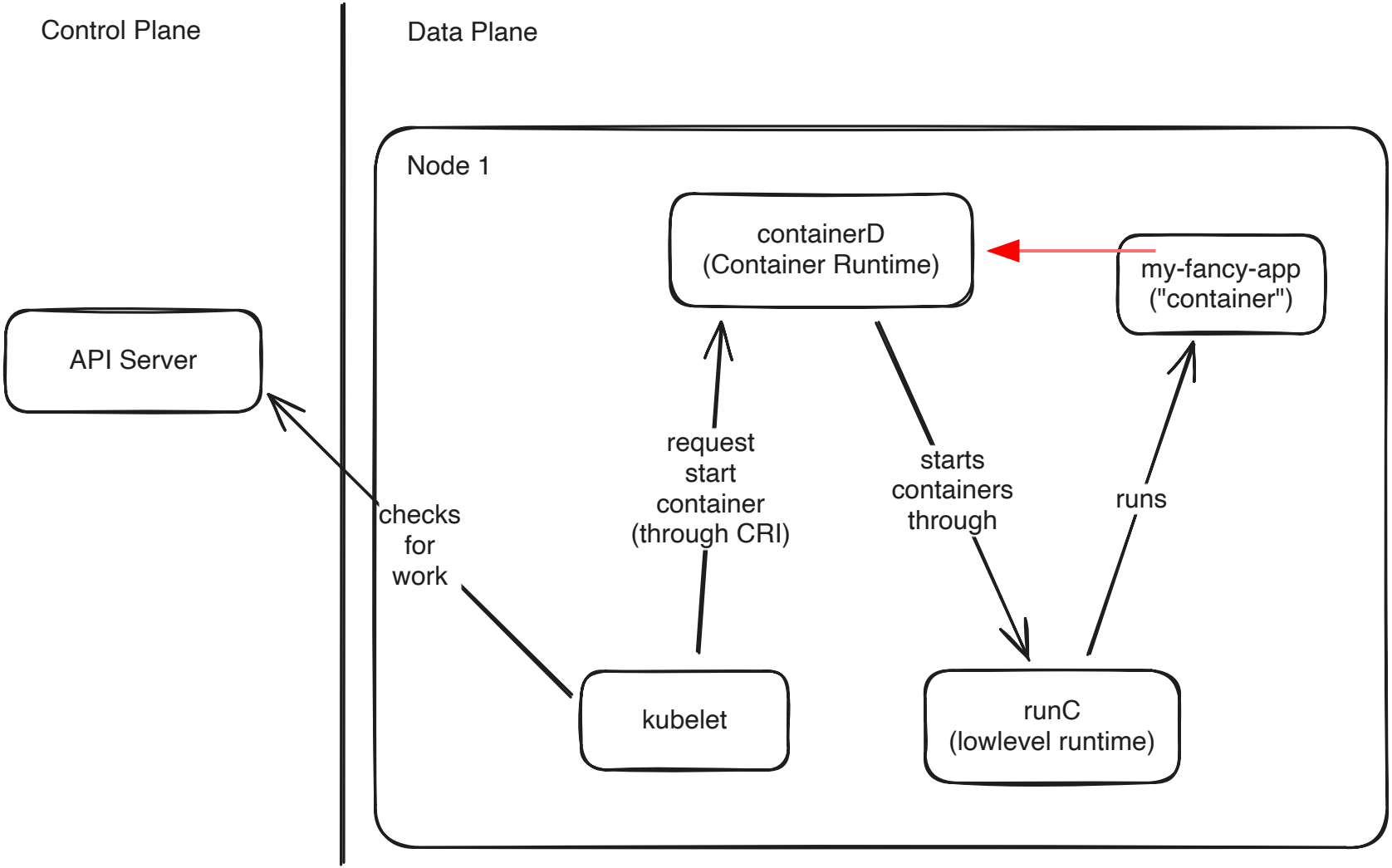

Control Plane
API Server
EtcD
Scheduler
Data Plane
Node 1
kubelet
my-fancy-app
Node 2
kubelet
awesome-app

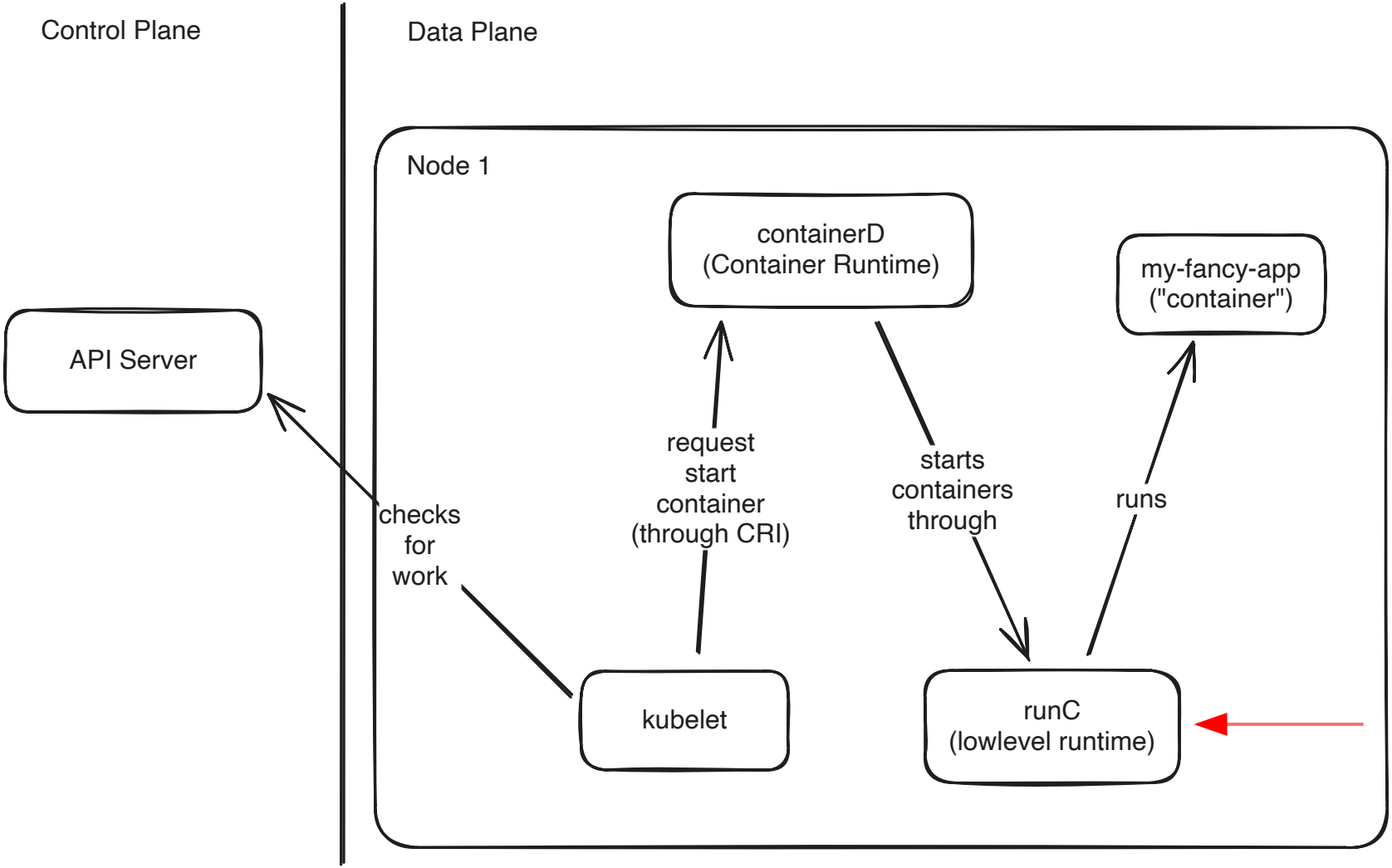

Control Plane
API Server
EtcD
Scheduler
Data Plane
Node 1
kubelet
my-fancy-app
Node 2
kubelet
awesome-app


Control Plane
API Server
EtcD
Scheduler
Data Plane
Node 1
kubelet
my-fancy-app
Node 2
kubelet
awesome-app


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)

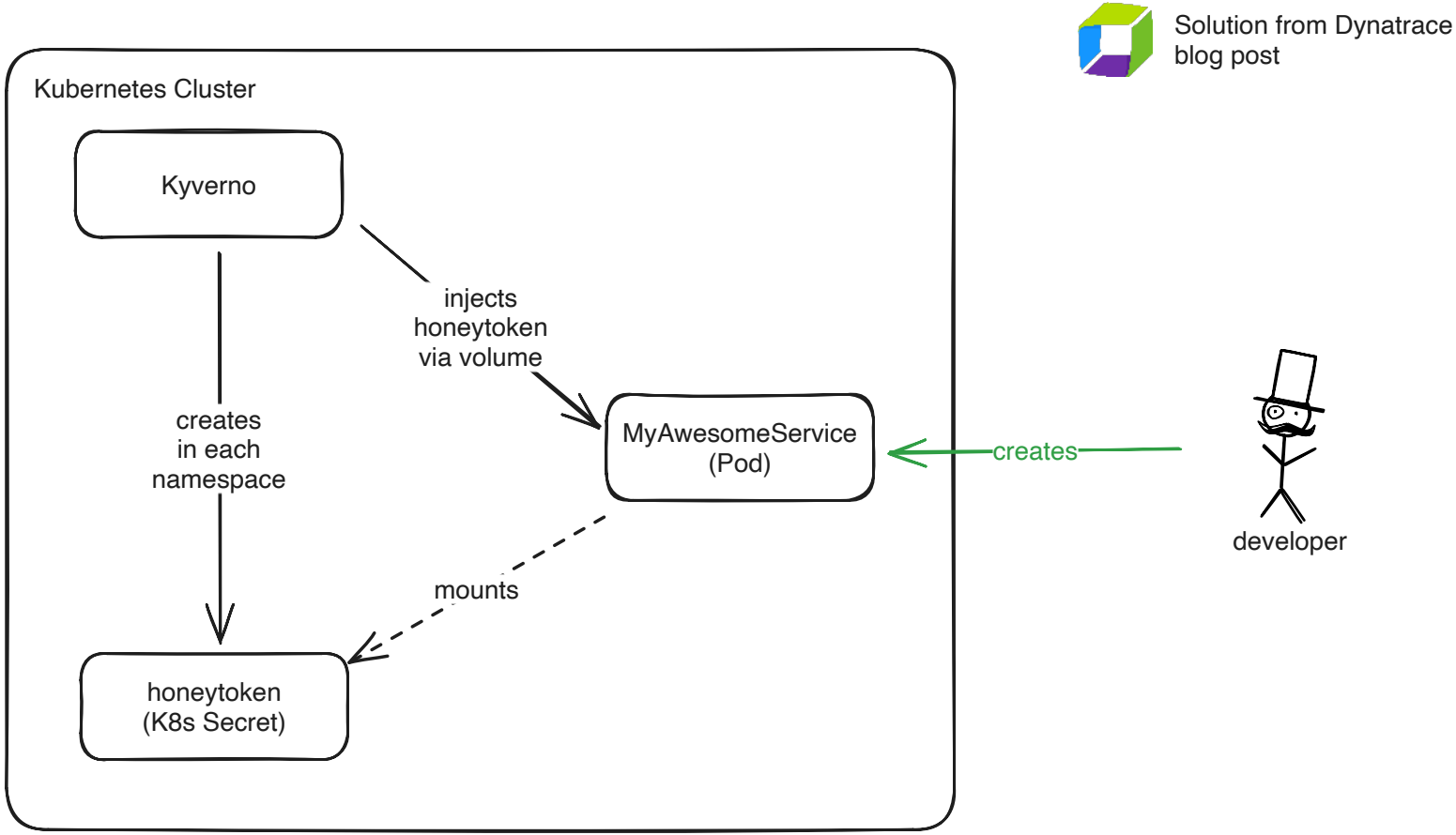
Cgroups

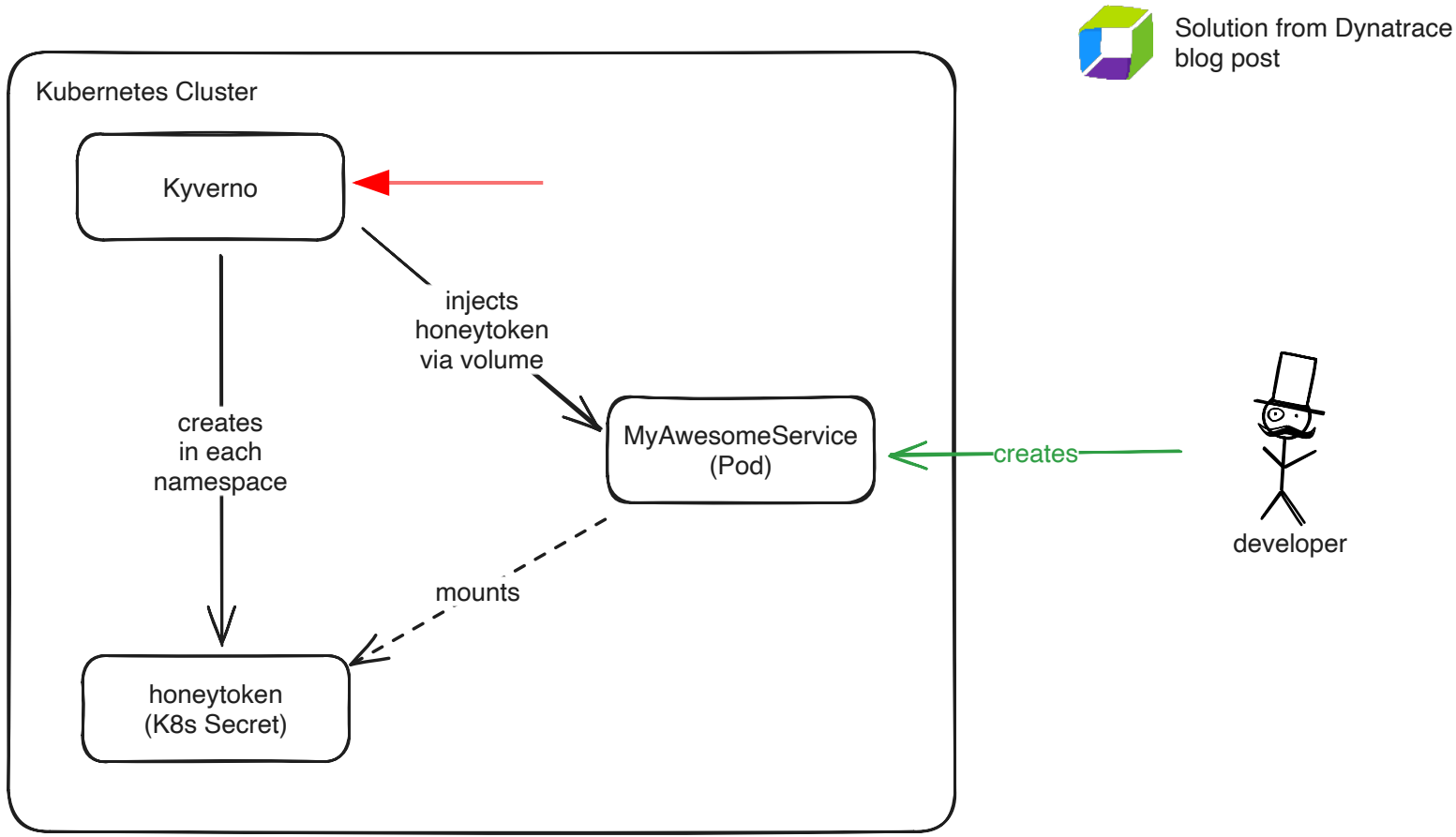
Limit resource usage (CPU, Memory)

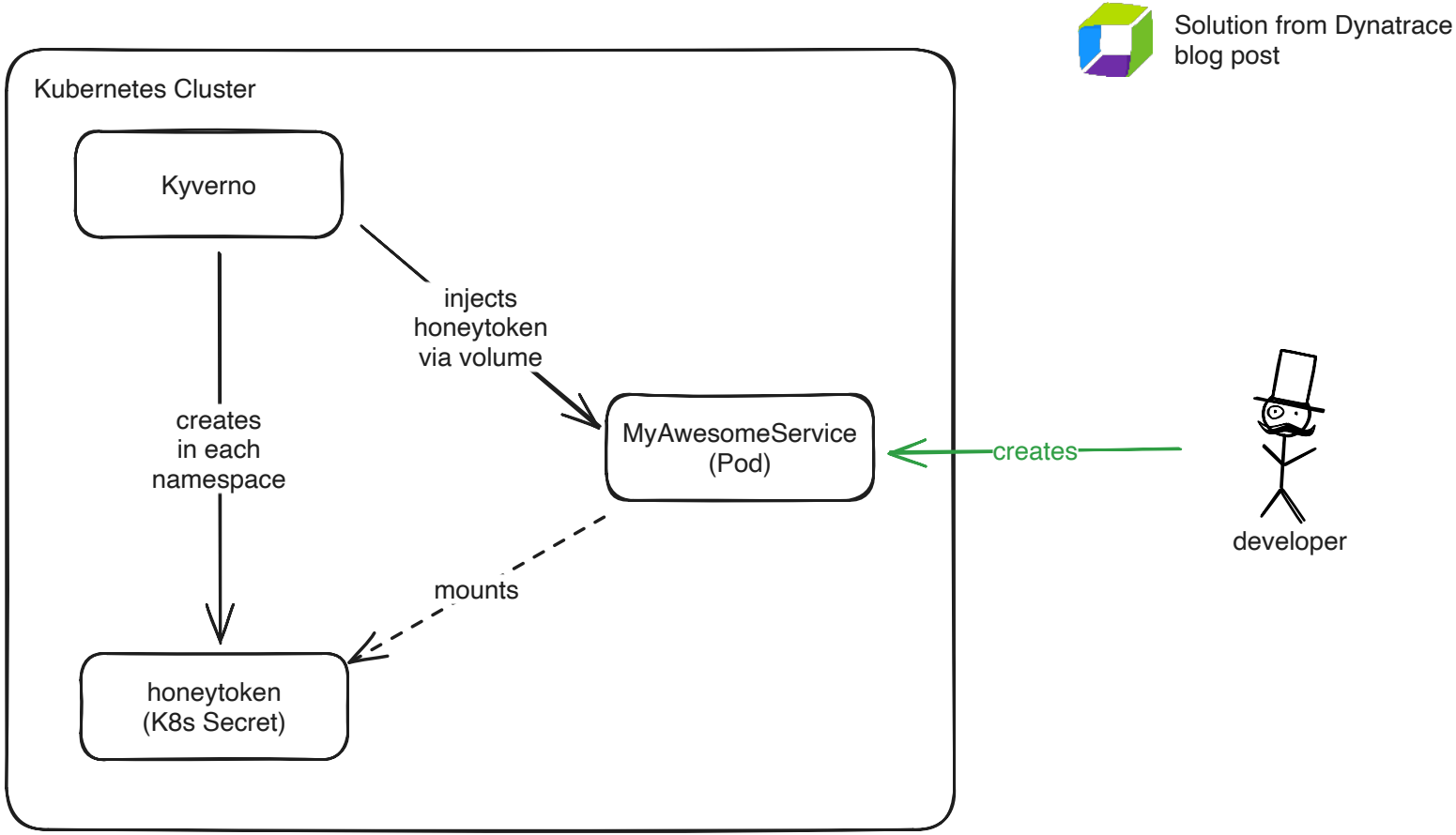
Namespaces

Separate areas (Networking, Mounts)

Image by wayhomestudio on Freepik


Solution from Dynatrace blog post
Kubernetes Cluster
Kyverno
injects honeytoken via volume
creates in each namespace
MyAwesomeService (Pod)
creates
developer
mounts
honeytoken (K8s Secret)


Solution from Dynatrace blog post
Kubernetes Cluster
Kyverno
injects honeytoken via volume
creates in each namespace
MyAwesomeService (Pod)
creates
developer
mounts
honeytoken (K8s Secret)


Solution from Dynatrace blog post
Kubernetes Cluster
Kyverno
injects honeytoken via volume
creates in each namespace
MyAwesomeService (Pod)
creates
developer
mounts
honeytoken (K8s Secret)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```

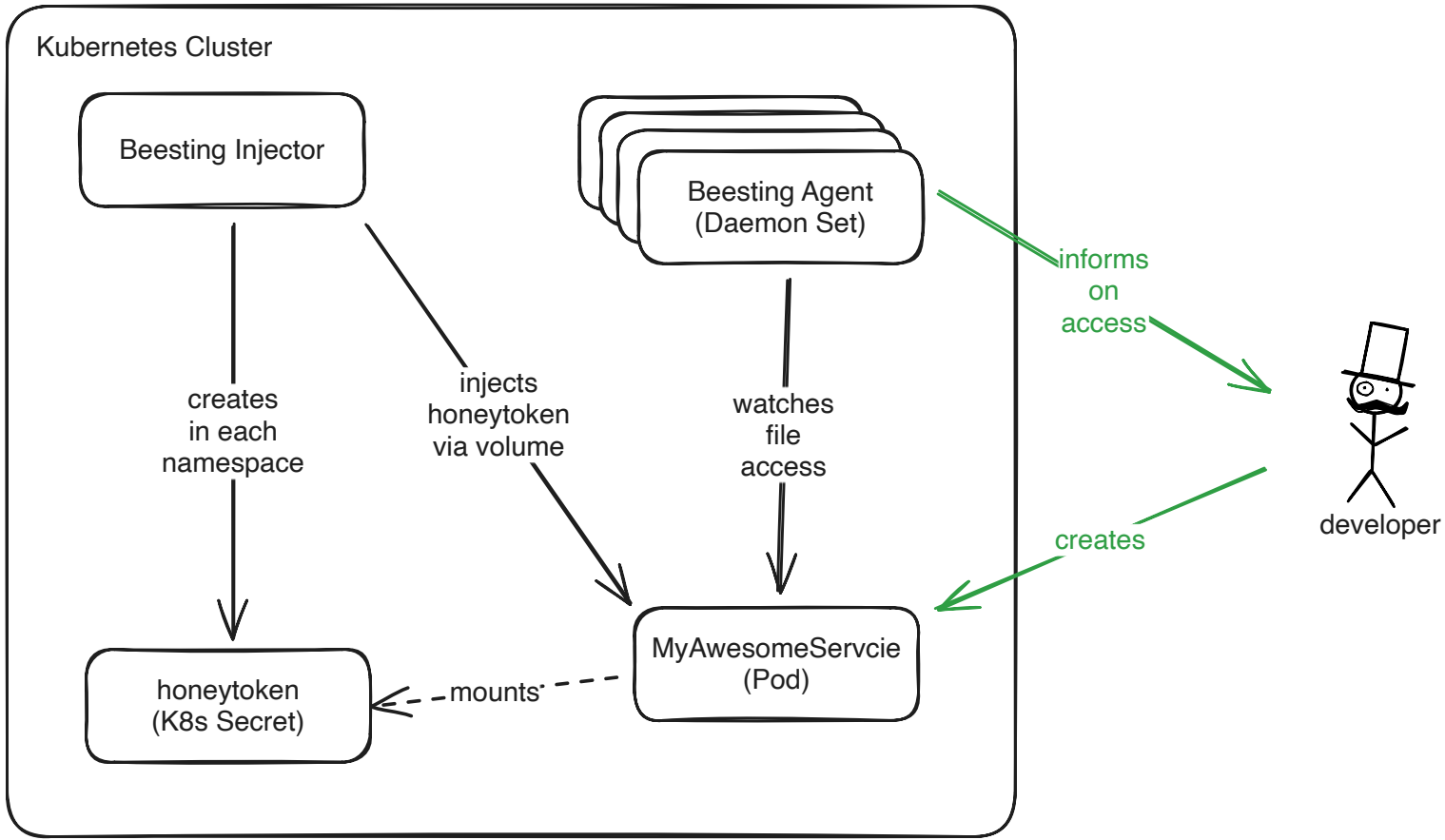
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: "myapp"
  namespace: default
spec:
  containers:
  - name: myapp
    image: "myapp:latest"
    volumeMounts:
    - name: honey-volume
      readOnly: true
      subPath: token
      mountPath: /run/secrets/eks.amazonaws.com/s3_token
  volumes:
    - name: honeytoken
      secret:
        secretName: honeytoken
```


Kubernetes Cluster
Beesting Injector
Beesting Agent
(Daemon Set)
creates
in each
namespace
injects
honeytoken
via volume
watches
file
access
informs
on
access
creates
developer
MyAwesomeServcie
(Pod)
mounts
honeytoken
(K8s Secret)

Problems

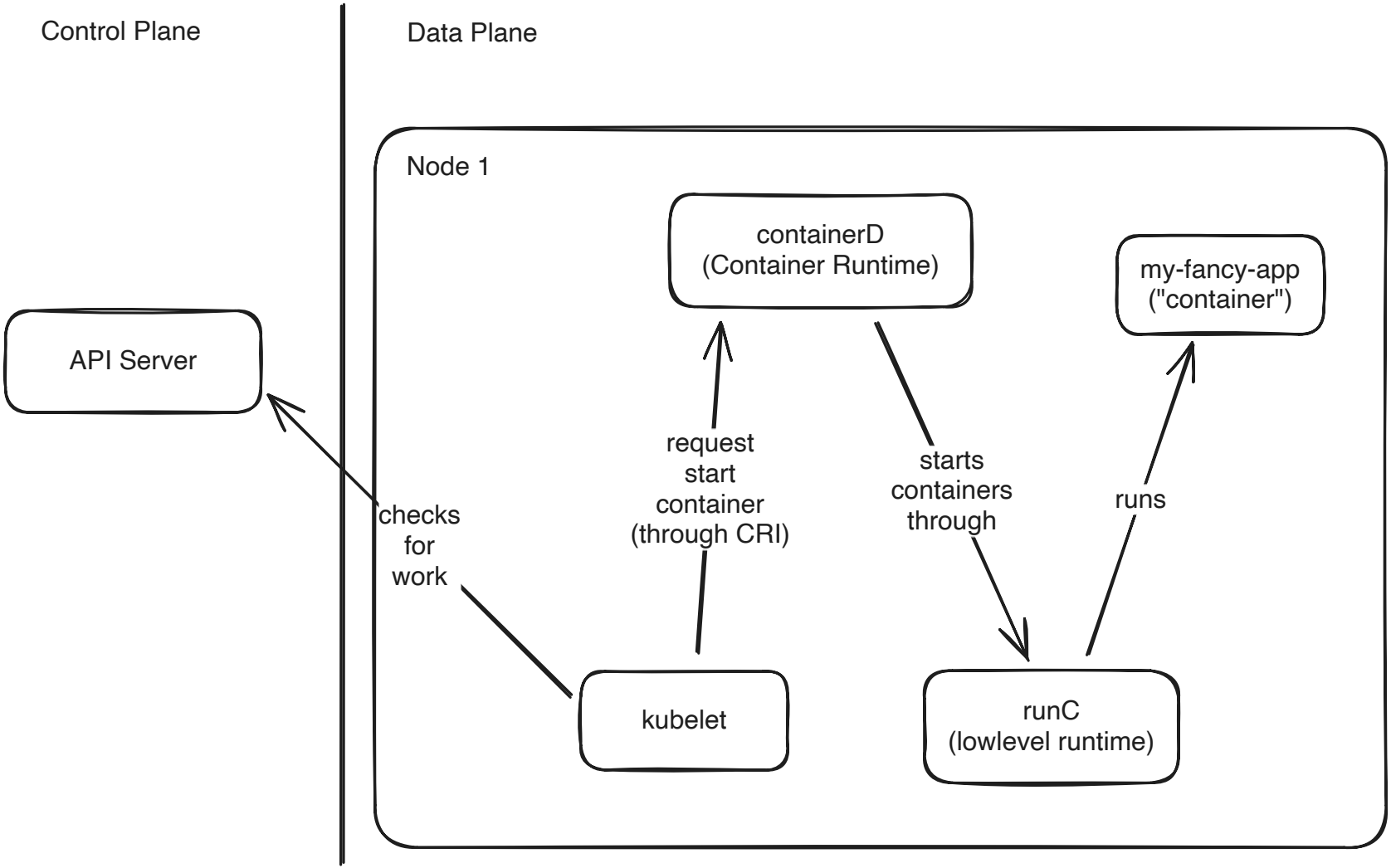
Problems

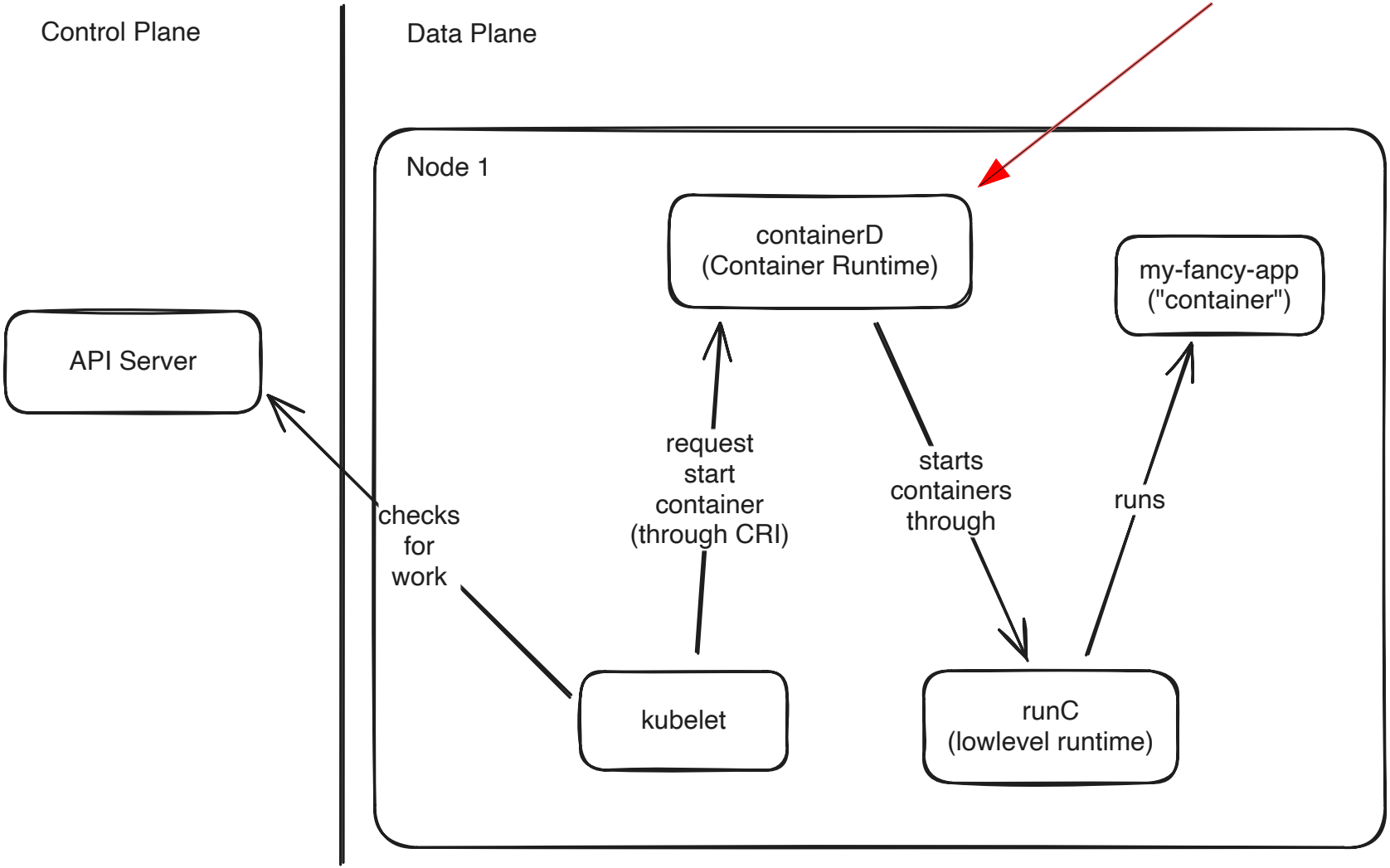
- Lot of moving parts

Problems

- Lot of moving parts
- Kubernetes specific

Boring


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)


Control Plane
Data Plane
Node 1
containerD
(Container Runtime)
my-fancy-app
("container")
API Server
request
start
container
(through CRI)
starts
containers
through
runs
checks
for
work
kubelet
runC
(lowlevel runtime)

main containerd / docs / PLUGINS.md

amalthundiyil docs: fix snapshots api import b78c5c6 · 2 months ago History

Preview Code Blame 296 lines (243 loc) · 11.1 KB Raw

containerd Plugins

containerd supports extending its functionality using most of its defined interfaces. This includes using a customized runtime, snapshotter, content store, and even adding gRPC interfaces.

Smart Client Model

containerd has a smart client architecture, meaning any functionality which is not required by the daemon is done by the client. This includes most high level interactions such as creating a container's specification, interacting with an image registry, or loading an image from tar. containerd's Go client gives a user access to many points of extensions from creating their own options on container creation to resolving image registry names.

See containerd's Go documentation

External Plugins

External plugins allow extending containerd's functionality using an officially released version of containerd without needing to recompile the daemon to add a plugin.

containerd allows extensions through two methods:

- via a binary available in containerd's PATH
- by configuring containerd to proxy to another gRPC service

V2 Runtimes

containerd supports multiple container runtimes. Each container can be invoked with a different runtime.

When using the Container Runtime Interface (CRI) plugin, named runtimes can be defined in the containerd configuration file. When a container is run without specifying a runtime, the configured default runtime is used. Alternatively, a different named runtime can be specified explicitly when creating a container via CRI gRPC by selecting the runtime handler to be used.

When a client such as `ctr` or `nerdctl` creates a container, it can optionally specify a runtime and options to use. If a runtime is not specified, containerd will use its default runtime.

containerd invokes v2 runtimes as binaries on the system, which are used to start the shim process for containerd. This, in turn, allows containerd to start and manage those containers using the runtime shim api returned by the binary.

For more details on runtimes and shims, including how to invoke and configure them, see the runtime v2 documentation

- Runtime

- Runtime
- Differ

- Runtime
- Differ
- and a few more

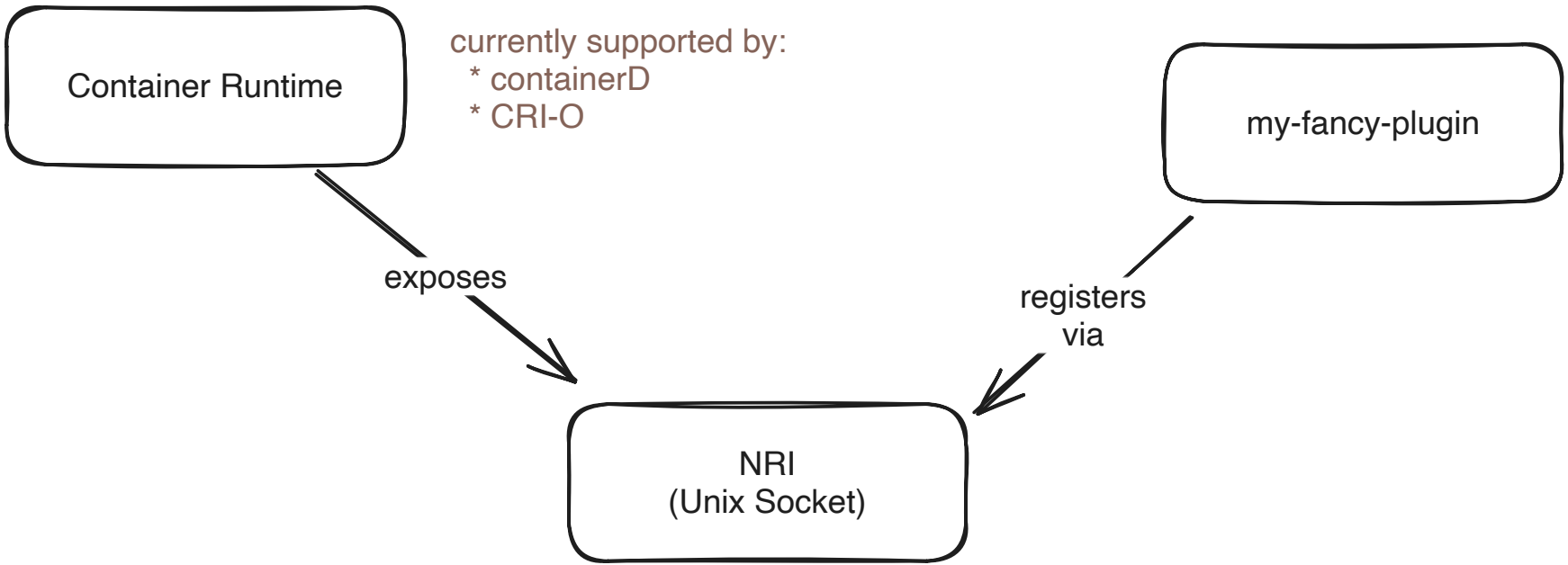
Differ Plugins

```
// Diff service creates and applies diffs
service Diff {
  // Apply applies the content associated with the provided digests onto
  // the provided mounts. Archive content will be extracted and
  // decompressed if necessary.
  rpc Apply(ApplyRequest) returns (ApplyResponse);

  // Diff creates a diff between the given mounts and uploads the result
  // to the content store.
  rpc Diff(DiffRequest) returns (DiffResponse);
}
```

```
// Diff service creates and applies diffs
service Diff {
  // Apply applies the content associated with the provided digests onto
  // the provided mounts. Archive content will be extracted and
  // decompressed if necessary.
  rpc Apply(ApplyRequest) returns (ApplyResponse);

  // Diff creates a diff between the given mounts and uploads the result
  // to the content store.
  rpc Diff(DiffRequest) returns (DiffResponse);
}
```

```
message DiffRequest {
  repeated containerd.types.Mount left = 1;
  repeated containerd.types.Mount right = 2;
  string media_type = 3;
  string ref = 4;
  map<string, string> labels = 5;
  google.protobuf.Timestamp source_date_epoch = 6;
}
```

Node Resource Interface


Container Runtime
currently supported by:
* containerD
* CRI-O
my-fancy-plugin
exposes
registers
via
NRI
(Unix Socket)

```go
type handlers struct {
  Configure           func(...) (api.EventMask, error)
  Synchronize         func(...) ([]*api.ContainerUpdate, error)
  Shutdown            func(...)
  RunPodSandbox       func(...) error
  StopPodSandbox      func(...) error
  RemovePodSandbox    func(...) error
  CreateContainer     func(...) (*api.ContainerAdjustment, []*api.ContainerUpdate, error)
  StartContainer      func(...) error
  UpdateContainer     func(...) ([]*api.ContainerUpdate, error)
  StopContainer       func(...) ([]*api.ContainerUpdate, error)
  RemoveContainer     func(...) error
  PostCreateContainer func(...) error
  PostStartContainer  func(...) error
  PostUpdateContainer func(...) error
}
```

```go
type handlers struct {
  Configure           func(...) (api.EventMask, error)
  Synchronize         func(...) ([]*api.ContainerUpdate, error)
  Shutdown            func(...)
  RunPodSandbox       func(...) error
  StopPodSandbox      func(...) error
  RemovePodSandbox    func(...) error
  CreateContainer     func(...) (*api.ContainerAdjustment, []*api.ContainerUpdate, error)
  StartContainer      func(...) error
  UpdateContainer     func(...) ([]*api.ContainerUpdate, error)
  StopContainer       func(...) ([]*api.ContainerUpdate, error)
  RemoveContainer     func(...) error
  PostCreateContainer func(...) error
  PostStartContainer  func(...) error
  PostUpdateContainer func(...) error
}
```

```go
type ContainerAdjustment struct {
  Annotations map[string]string
  Mounts      []*Mount
  Env         []*KeyValue
  Hooks       *Hooks
  Linux       *LinuxContainerAdjustment
  Rlimits     []*POSIXRlimit
  CDIDevices  []*CDIDevice
}
```

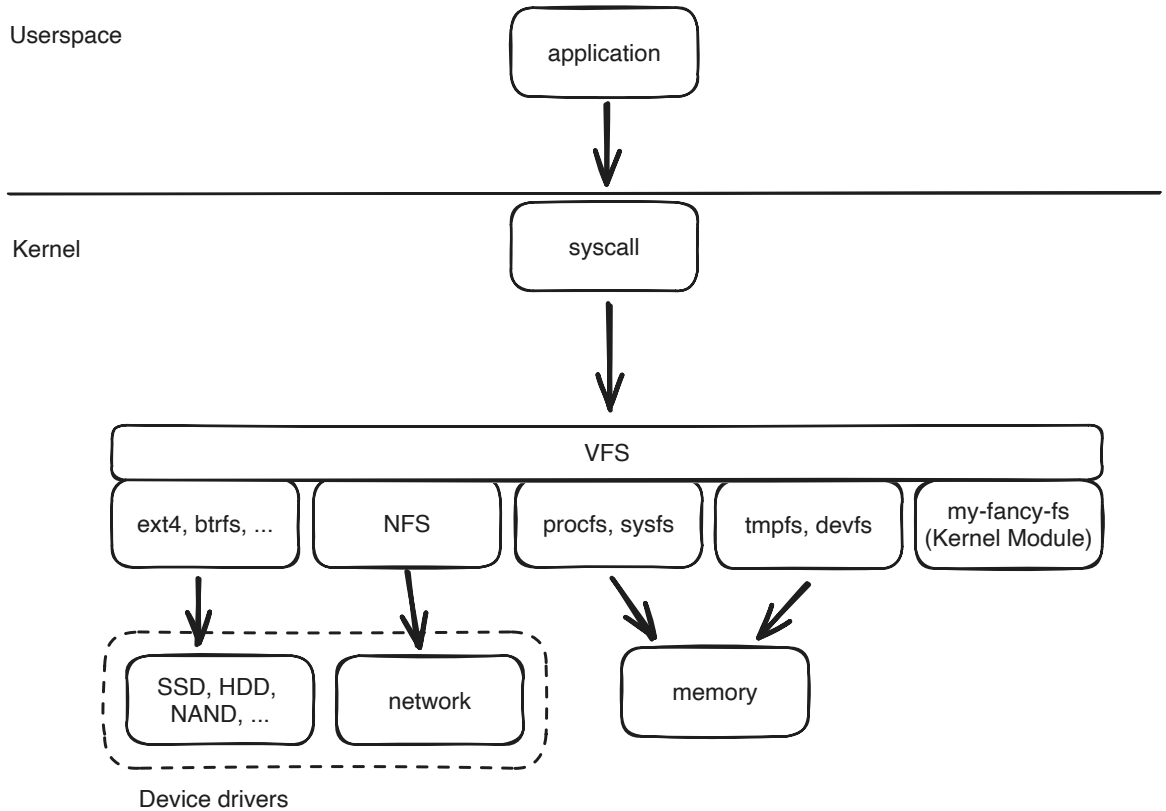
```go
type ContainerAdjustment struct {
  Annotations map[string]string
  Mounts      []*Mount
  Env         []*KeyValue
  Hooks       *Hooks
  Linux       *LinuxContainerAdjustment
  Rlimits     []*POSIXRlimit
  CDIDevices  []*CDIDevice
}
```

```go
type ContainerAdjustment struct {
  Annotations map[string]string
  Mounts      []*Mount
  Env         []*KeyValue
  Hooks       *Hooks
  Linux       *LinuxContainerAdjustment
  Rlimits     []*POSIXRlimit
  CDIDevices  []*CDIDevice
}
```

What to inject into the container?

Honeytokens = Files

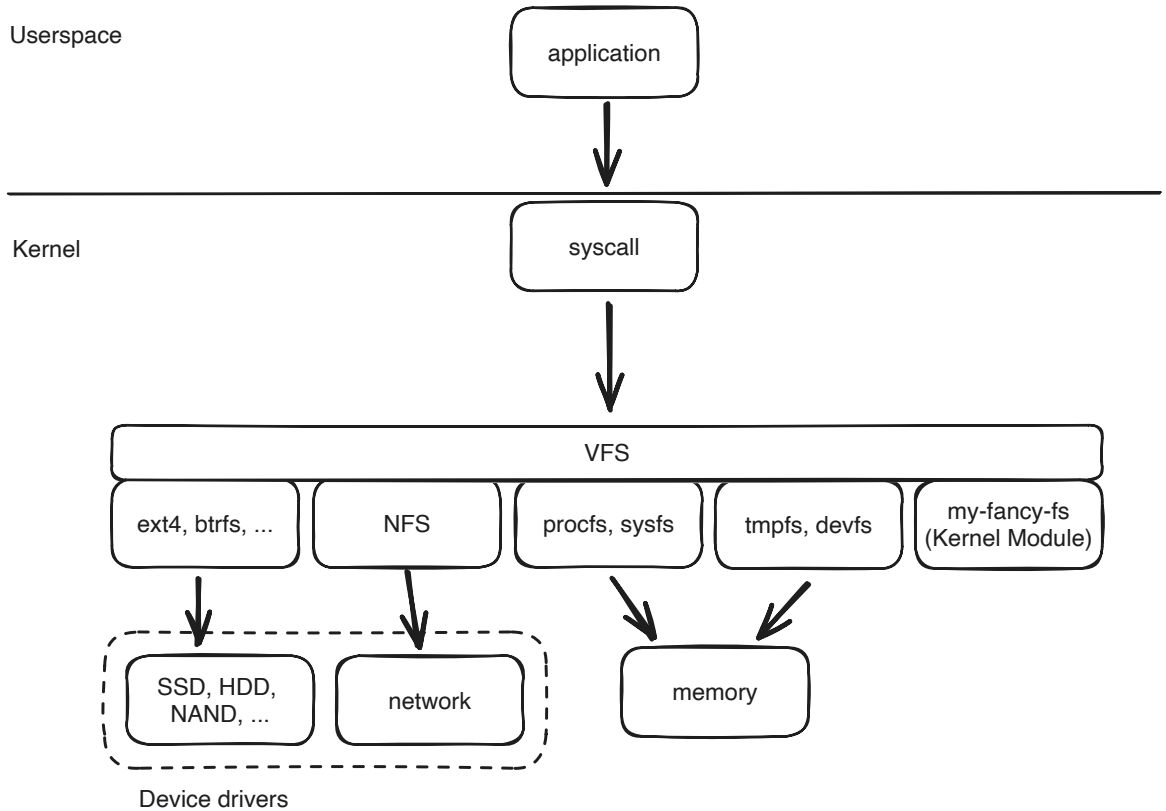
- ext2, ext3, ext3
- BTRFS
- XFS
- ZFS
- and many more


Userspace
application
Kernel
syscall
VFS
ext4, btrfs, ...
NFS
procfs, sysfs
tmpfs, devfs
my-fancy-fs
(Kernel Module)
SSD, HDD,
NAND, ...
network
memory
Device drivers

What if ...

we create our own

File System?


Userspace
application
Kernel
syscall
VFS
ext4, btrfs, ...
NFS
procfs, sysfs
tmpfs, devfs
my-fancy-fs
(Kernel Module)
SSD, HDD,
NAND, ...
network
memory
Device drivers

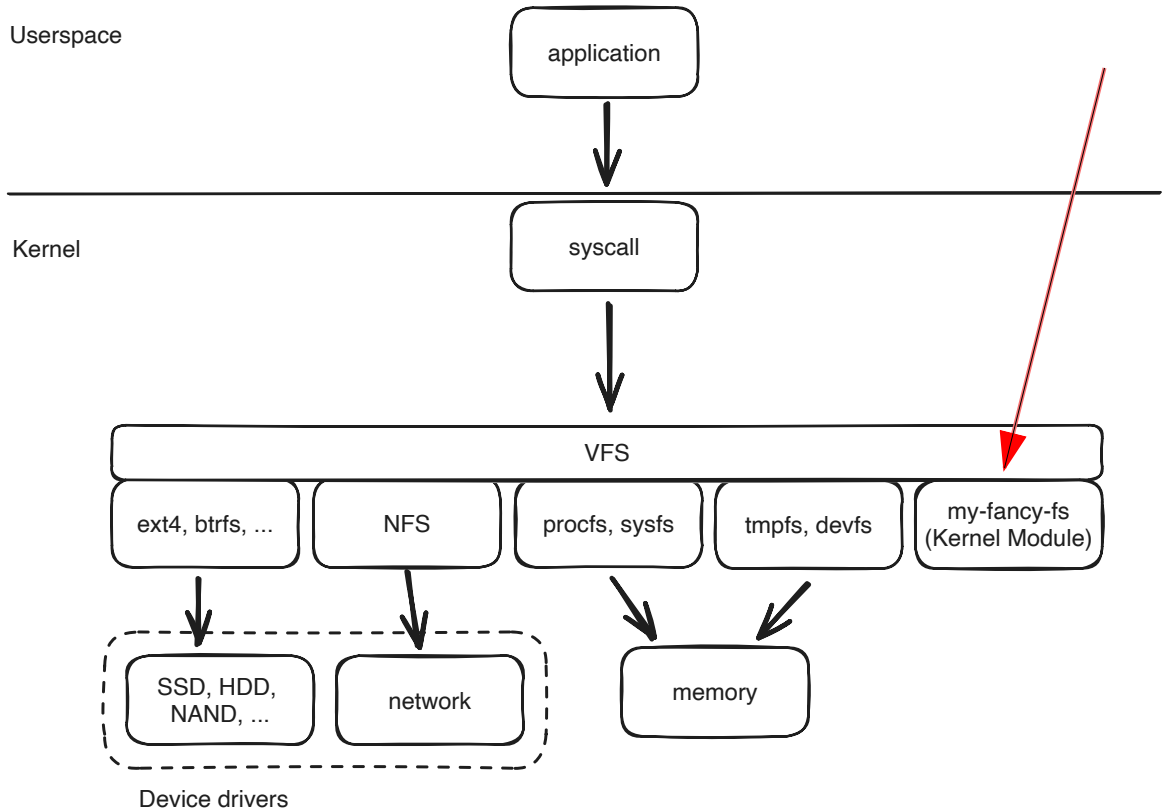
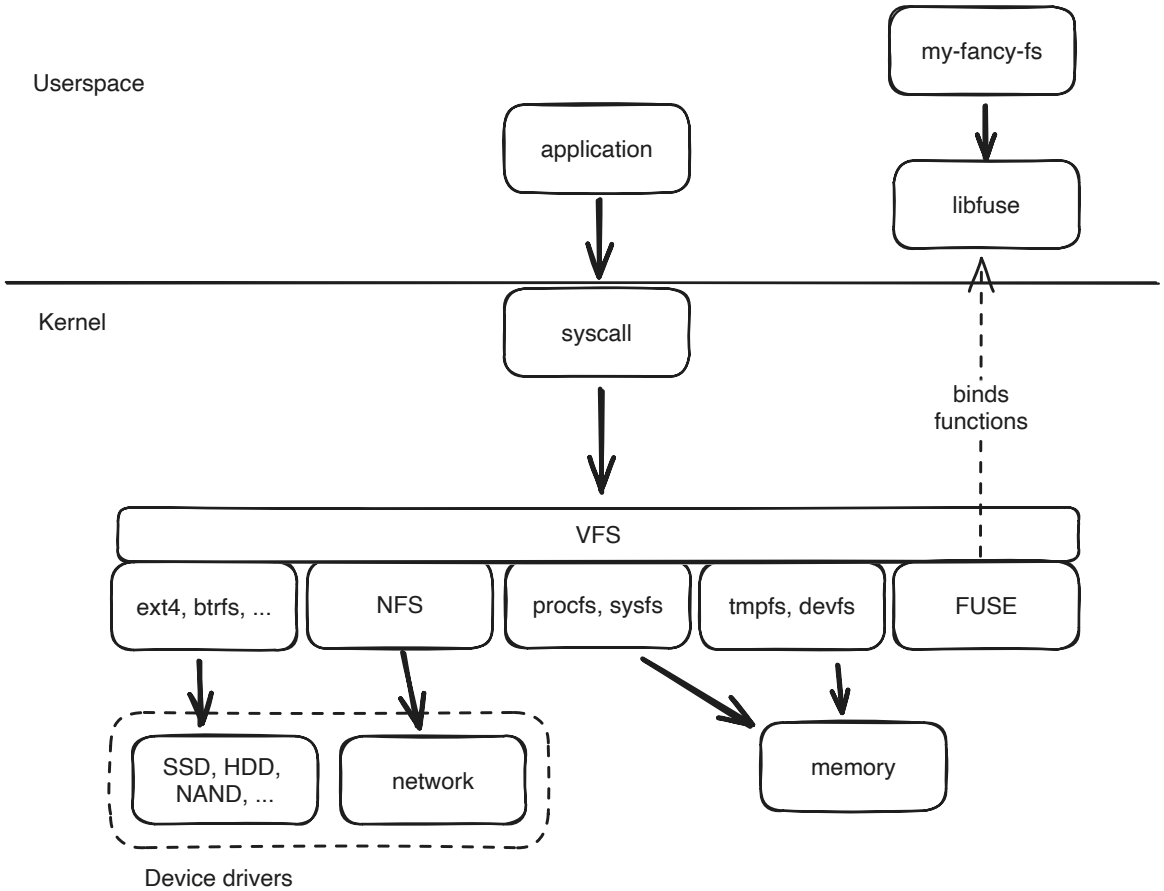
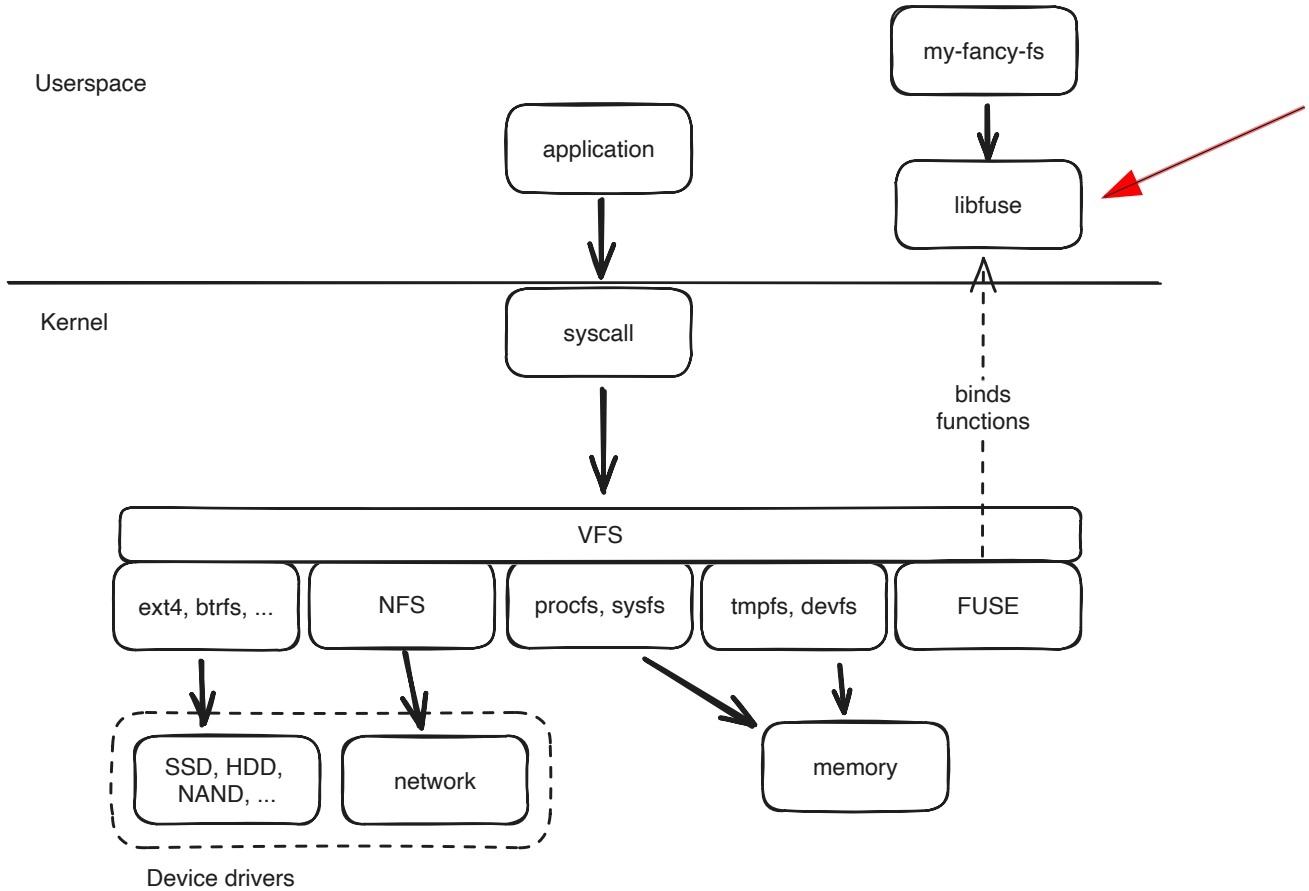

Userspace
application
Kernel
syscall
VFS
ext4, btrfs, ...
NFS
procfs, sysfs
tmpfs, devfs
my-fancy-fs
(Kernel Module)
SSD, HDD,
NAND, ...
network
memory
Device drivers

Image by cookie_studio on pixabay


Userspace
my-fancy-fs
application
libfuse
Kernel
syscall
binds
functions
VFS
ext4, btrfs, ...
NFS
procfs, sysfs
tmpfs, devfs
FUSE
SSD, HDD,
NAND, ...
network
memory
Device drivers


Userspace
my-fancy-fs
application
libfuse
Kernel
syscall
binds
functions
VFS
ext4, btrfs, ...
NFS
procfs, sysfs
tmpfs, devfs
FUSE
SSD, HDD,
NAND, ...
network
memory
Device drivers

What about OCI Hooks?

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hook struct {
  Path    string
  Args    []string
  Env     []string
  Timeout *OptionalInt
}
```

```go
type Hooks struct {
  Prestart        []*Hook
  CreateRuntime   []*Hook
  CreateContainer []*Hook
  StartContainer  []*Hook
  Poststart       []*Hook
  Poststop        []*Hook
}
```

```go
type Hooks struct {
  Prestart        []*Hook
  CreateRuntime   []*Hook
  CreateContainer []*Hook
  StartContainer  []*Hook
  Poststart       []*Hook
  Poststop        []*Hook
}
```

v1.2.0 runtime-spec / config.md ↑ Top

Preview Code Blame 1160 lines (1024 loc) · 54.4 KB · Raw

CreateContainer Hooks

The `createContainer` hooks MUST be called as part of the `create` operation after the runtime environment has been created (according to the configuration in config.json) but before the `pivot_root` or any equivalent operation has been executed. The `createContainer` hooks MUST be called after the `createRuntime` hooks.

The `createContainer` hooks' path MUST resolve in the runtime namespace. The `createContainer` hooks MUST be executed in the container namespace.

For example, on Linux this would happen before the `pivot_root` operation is executed but after the mount namespace was created and setup.

The definition of `createContainer` hooks is currently underspecified and hooks authors, should only expect from the runtime that the mount namespace and different mounts will be setup. Other operations such as cgroups and SELinux/AppArmor labels might not have been performed by the runtime.

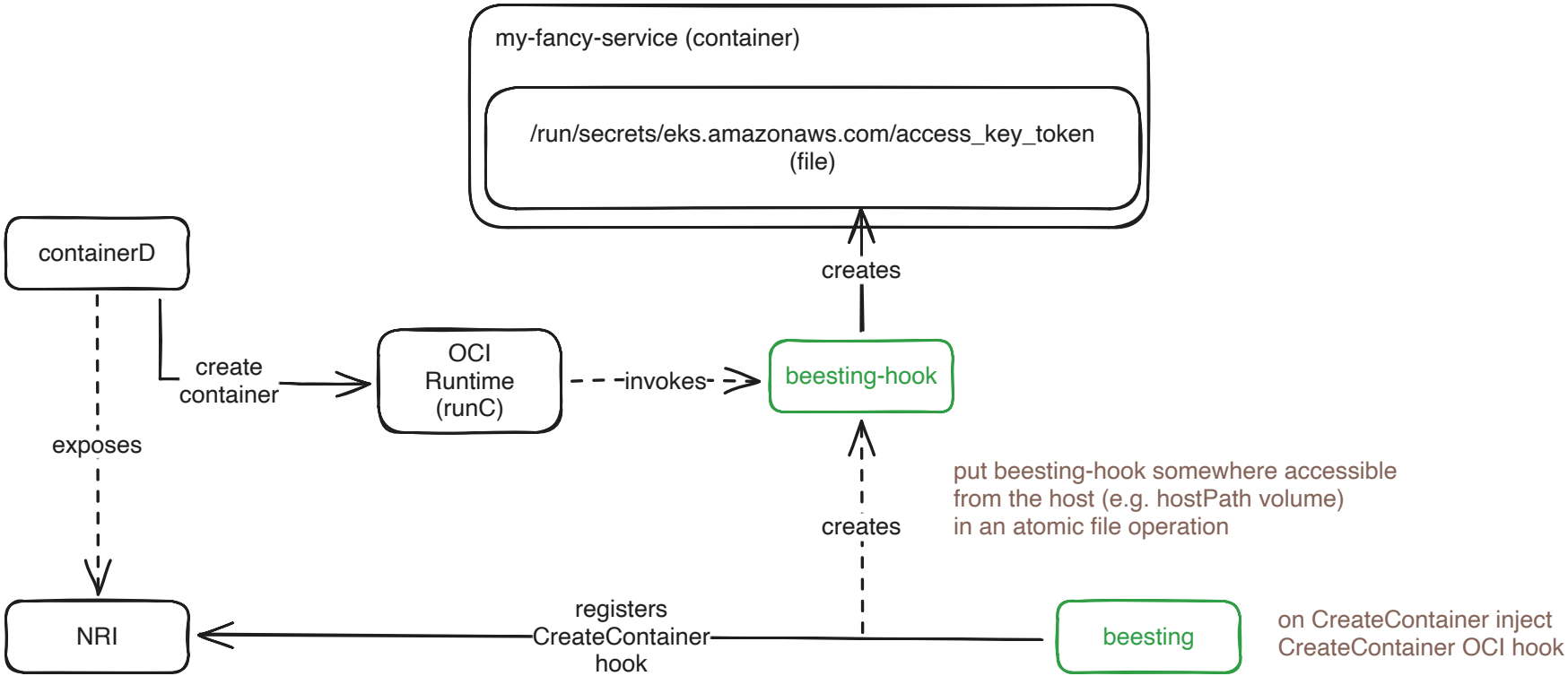
StartContainer Hooks

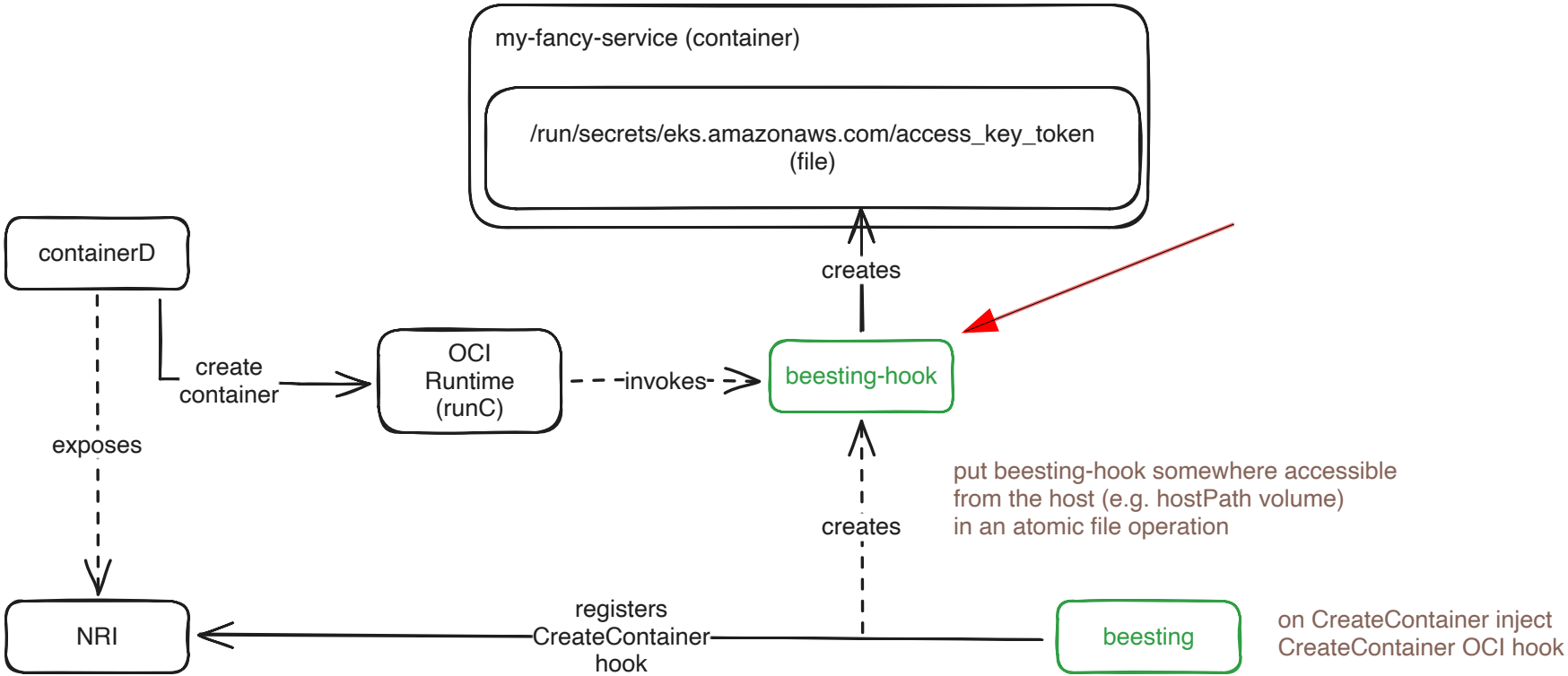
The `startContainer` hooks MUST be called before the user-specified process is executed as part of the `start` operation. This hook can be used to execute some operations in the container, for example running the `ldconfig` binary on linux before the container process is spawned.

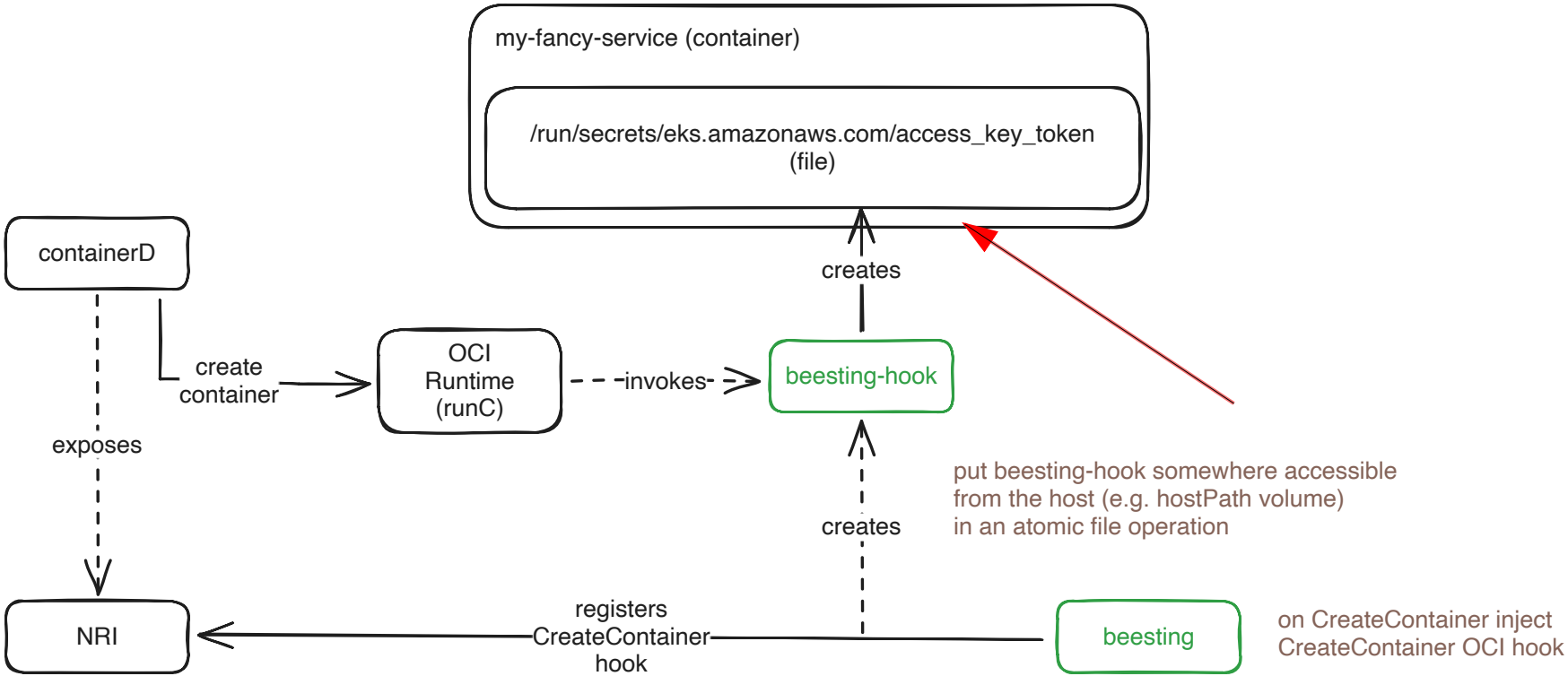
The `startContainer` hooks' path MUST resolve in the container namespace. The `startContainer` hooks MUST be executed in the container namespace.

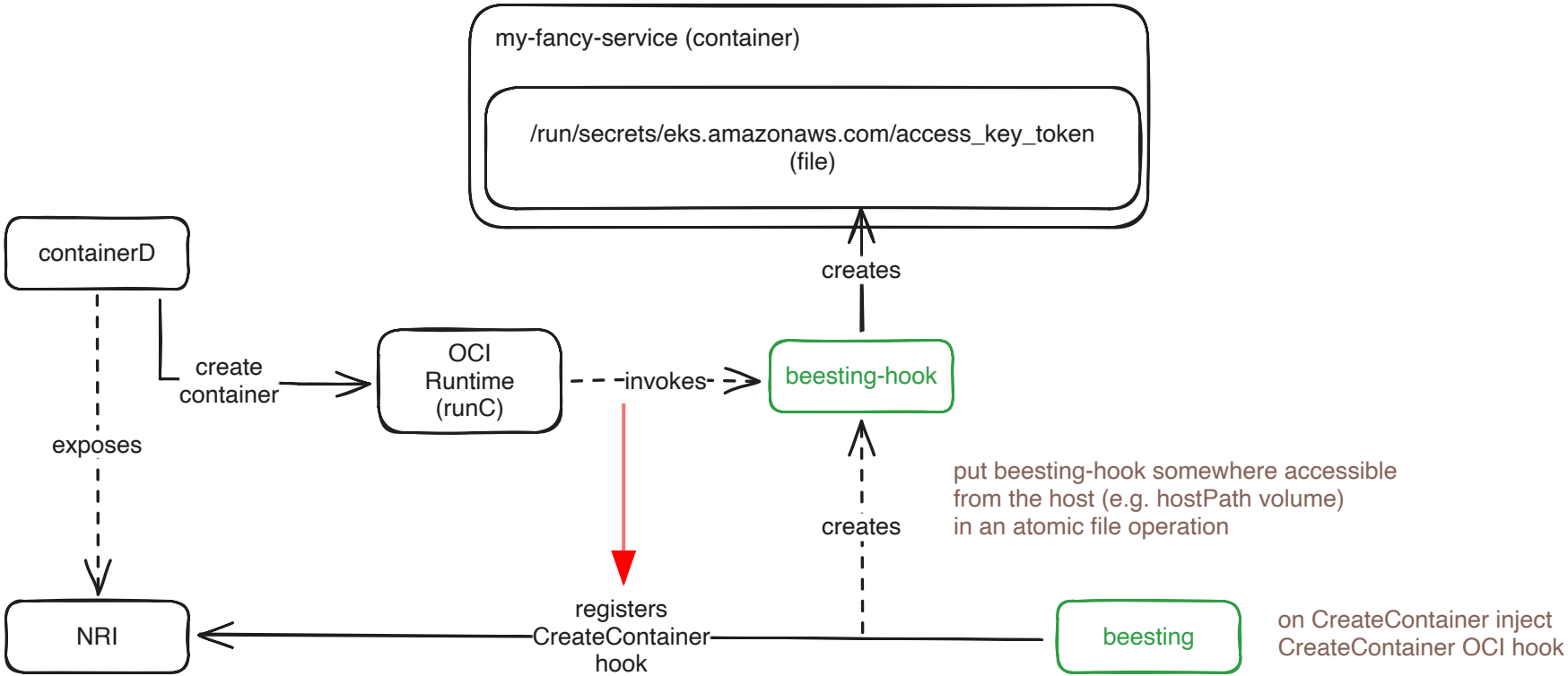
Link to docs

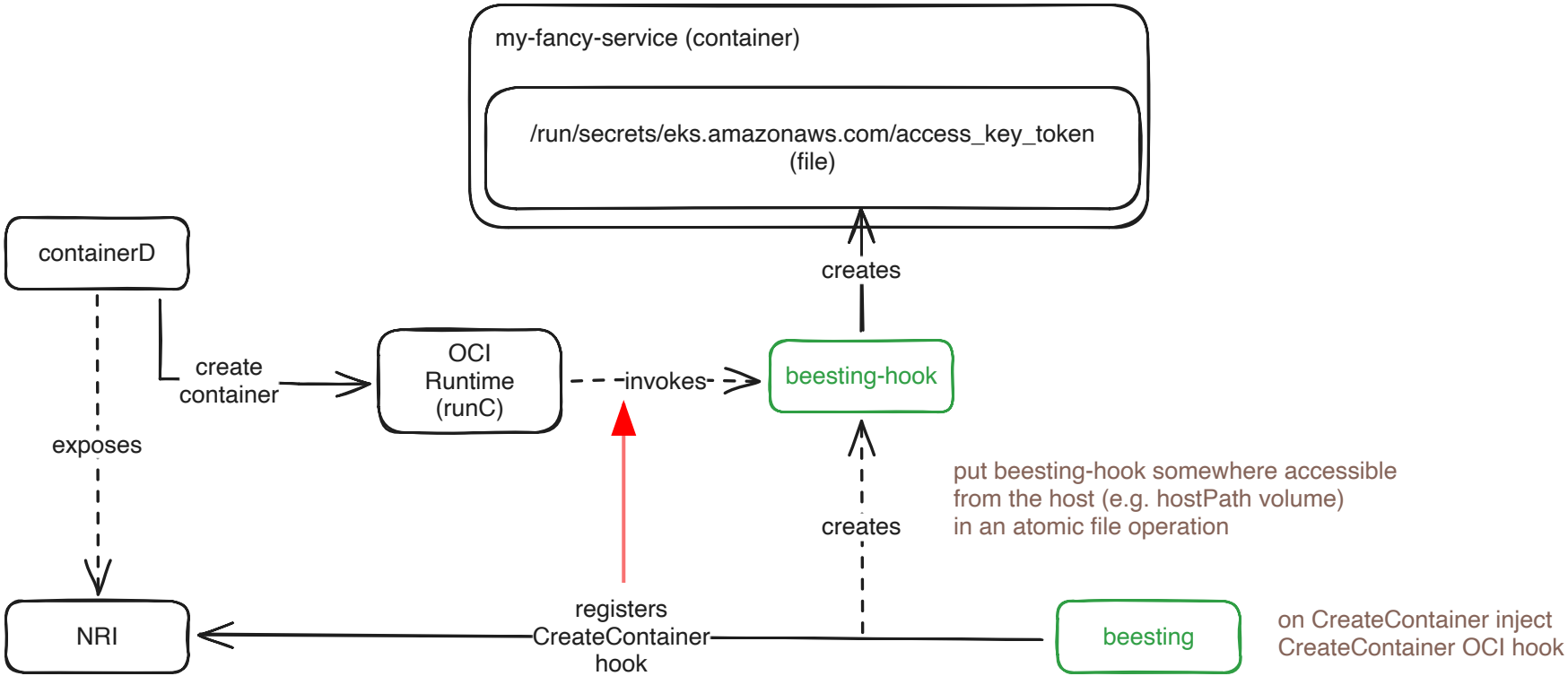
For now just file injection

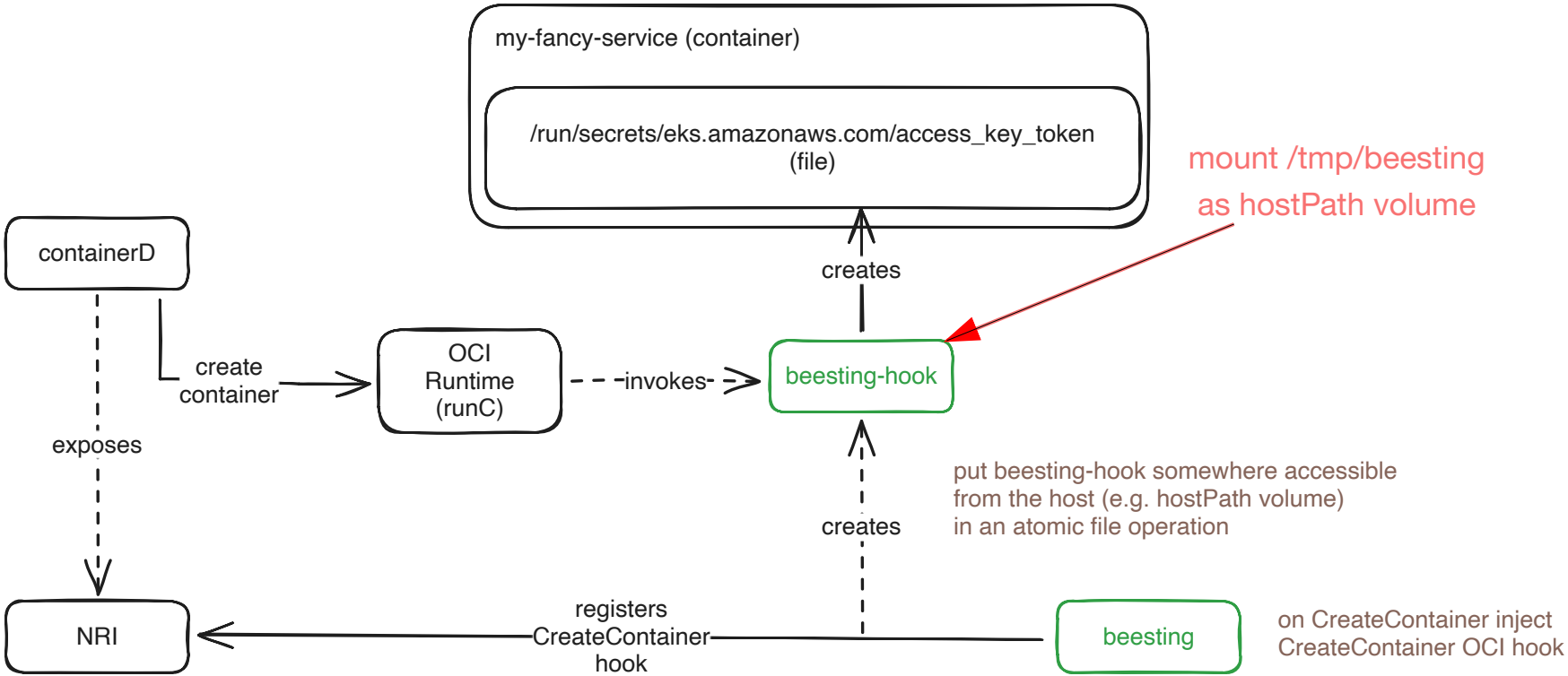

my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
registers
CreateContainer
hook
NRI
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
registers
CreateContainer
hook
NRI
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
mount /tmp/beesting
as hostPath volume
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook

beesting-hook is embedded into Beesting

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs
```

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created
```

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                        READY   STATUS              RESTARTS     AGE
beesting-agent-q9s2d        1/1     Running             0            54s
dummy-8984df79-zpvnm        0/1     RunContainerError   2 (4s ago)   20s
```

```
$ k describe pod dummy-8984df79-zpvnm
Name:             dummy-8984df79-zpvnm
Namespace:        default
Priority:         0
Service Account:  default
Node:             beestinger-control-plane/172.18.0.3
...
Events:
Type    Reason  Age                From     Message
----    ------  ----               ----     -------
...
Warning Failed  8s (x4 over 49s) kubelet  Error:   failed  to   create  containerd
                                          task:  failed to  create shim  task: OCI
                                          runtime   create  failed:   runc  create
                                          failed:   unable   to  start   container
                                          process:  error  during container  init:
                                          error   running   hook   #1:   fork/exec
                                          /tmp/beesting/beesting-hook:
                                          permission denied: unknown
```

```
$ k describe pod dummy-8984df79-zpvnm
Name:             dummy-8984df79-zpvnm
Namespace:        default
Priority:         0
Service Account:  default
Node:             beestinger-control-plane/172.18.0.3
...
Events:
Type    Reason  Age                From     Message
----    ------  ----               ----     -------
...
Warning Failed  8s (x4 over 49s) kubelet  Error:   failed  to   create  containerd
                                          task:  failed to  create shim  task: OCI
                                          runtime   create  failed:   runc  create
                                          failed:   unable   to  start   container
                                          process:  error  during container  init:
                                          error   running   hook   #1:   fork/exec
                                          /tmp/beesting/beesting-hook:
                                          permission denied: unknown
```

```
Error: failed to create containerd task: failed to create shim task:
OCI runtime create failed: runc create failed: unable to start container
process: error during container init: error running hook #1: fork/exec
/tmp/beesting/beesting-hook: permission denied: unknown
```

```
$ root@beestinger-control-plane:/# mount
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime)
...
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k,inode64)
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,noexec,relatime,inode64)
/dev/vda1 on /var type ext4 (rw,relatime,discard,errors=remount-ro)
devpts on /dev/console type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
...
```

```
$ root@beestinger-control-plane:/# mount
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime)
...
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k,inode64)
tmpfs on /tmp type tmpfs (rw,nosuid,nodev,noexec,relatime,inode64)
/dev/vda1 on /var type ext4 (rw,relatime,discard,errors=remount-ro)
devpts on /dev/console type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
...
```

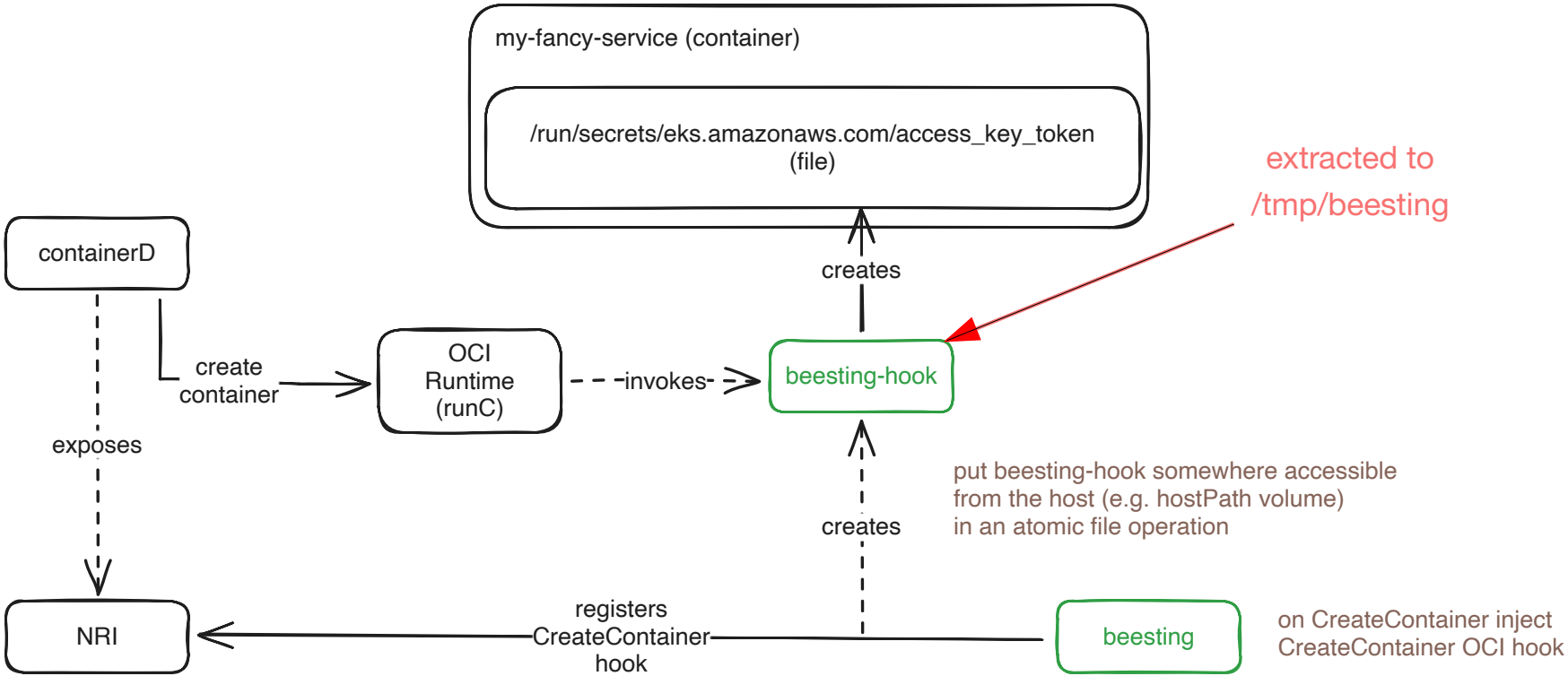
1.24.2 istio / cni / pkg / install / install.go ↑ Top

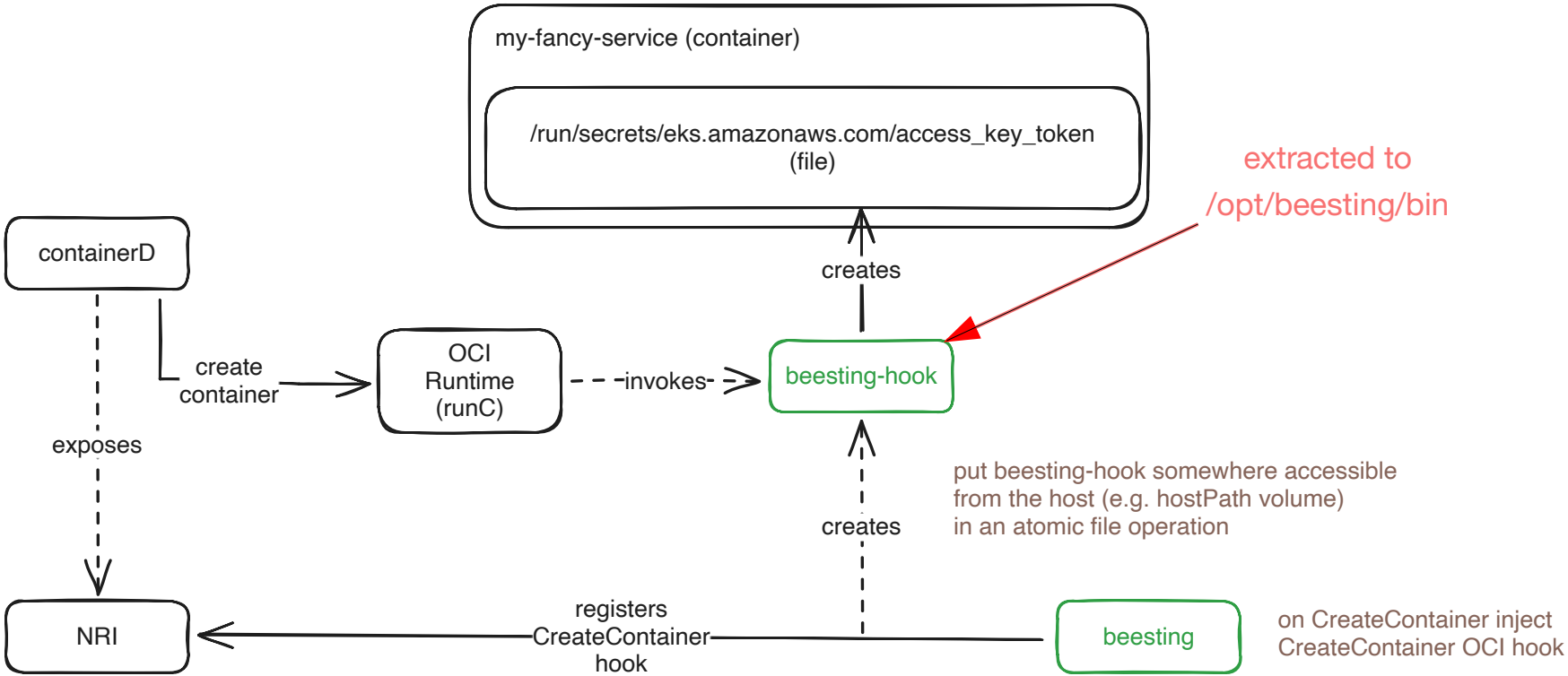
Code Blame 361 lines (331 loc) · 13.5 KB · Raw

```go
type Installer struct {
func NewInstaller(cfg *config.InstallConfig, isReady *atomic.Value) *Installer {
        return &Installer{
                cfg:                cfg,
                kubeconfigFilepath: filepath.Join(cfg.CNIAgentRunDir, constants.CNIPluginKubeconfName),
                isReady:            isReady,
        }
}

func (in *Installer) installAll(ctx context.Context) (sets.String, error) {
        // Install binaries
        // Currently we _always_ do this, since the binaries do not live in a shared location
        // and we harm no one by doing so.
        copiedFiles, err := copyBinaries(in.cfg.CNIBinSourceDir, in.cfg.CNIBinTargetDirs)
        if err != nil {
                cniInstalls.With(resultLabel.Value(resultCopyBinariesFailure)).Increment()
                return copiedFiles, fmt.Errorf("copy binaries: %v", err)
        }

        // Write kubeconfig with our current service account token as the contents, to the Istio agent rundir.
        // We do not write this to the common/shared CNI config dir, because it's not CNI config, we do not
        // need to watch it, and writing non-shared stuff to that location creates churn for other node agents.
        // Only our plugin consumes this kubeconfig, and it resides in our owned rundir on the host node,
        // so we are good to simply write it out if our watched svcacct token changes.
        if err := writeKubeConfigFile(in.cfg); err != nil {
                cniInstalls.With(resultLabel.Value(resultCreateKubeConfigFailure)).Increment()
                return copiedFiles, fmt.Errorf("write kubeconfig: %v", err)
        }

        // Install CNI netdir config (if needed) - we write/update this in the shared node CNI netdir,
        // which may be watched by other CNIs, and so we don't want to trigger writes to this file
        // unless it's missing or the contents are not what we expect.
        if err := checkValidCNIConfig(in.cfg, in.cniConfigFilepath); err != nil {
                installLog.Infof("configuration requires updates, (re)writing CNI config file at %q: %v", in.cniConfigFilepath, err)
```

/opt/cni/bin

cni/pkg/install/install.go


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
extracted to
/tmp/beesting
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
extracted to
/opt/beesting/bin
containerD
creates
create
container
OCI
Runtime
(runC)
invokes
beesting-hook
exposes
put beesting-hook somewhere accessible
from the host (e.g. hostPath volume)
in an atomic file operation
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
CreateContainer OCI hook

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                        READY   STATUS              RESTARTS     AGE
beesting-agent-q9s2d        1/1     Running             0            30s
dummy-8984df79-zpvnm        1/1     Running             0            52s
```

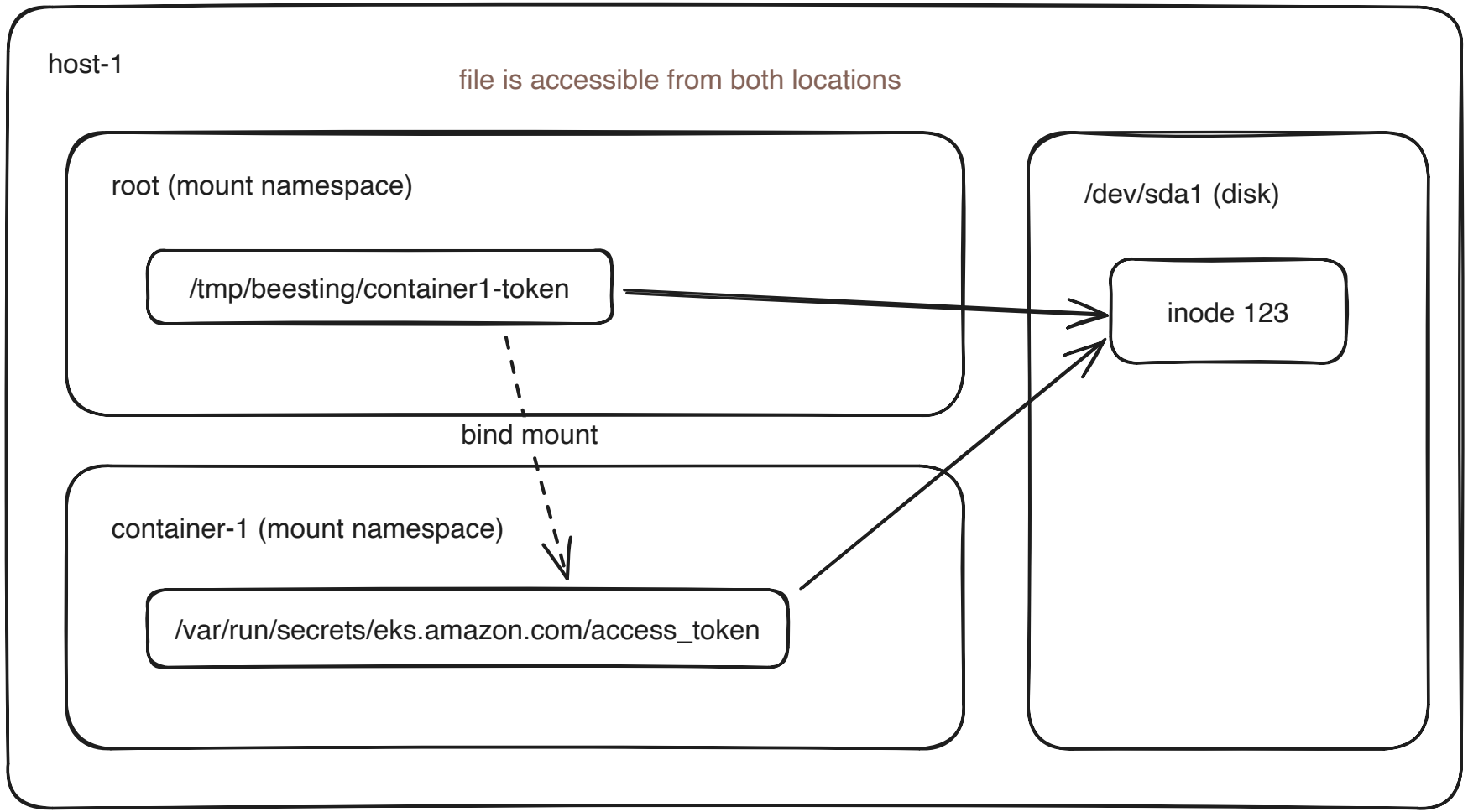
```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                        READY   STATUS              RESTARTS     AGE
beesting-agent-q9s2d        1/1     Running             0            30s
dummy-8984df79-zpvnm        1/1     Running             0            52s
```

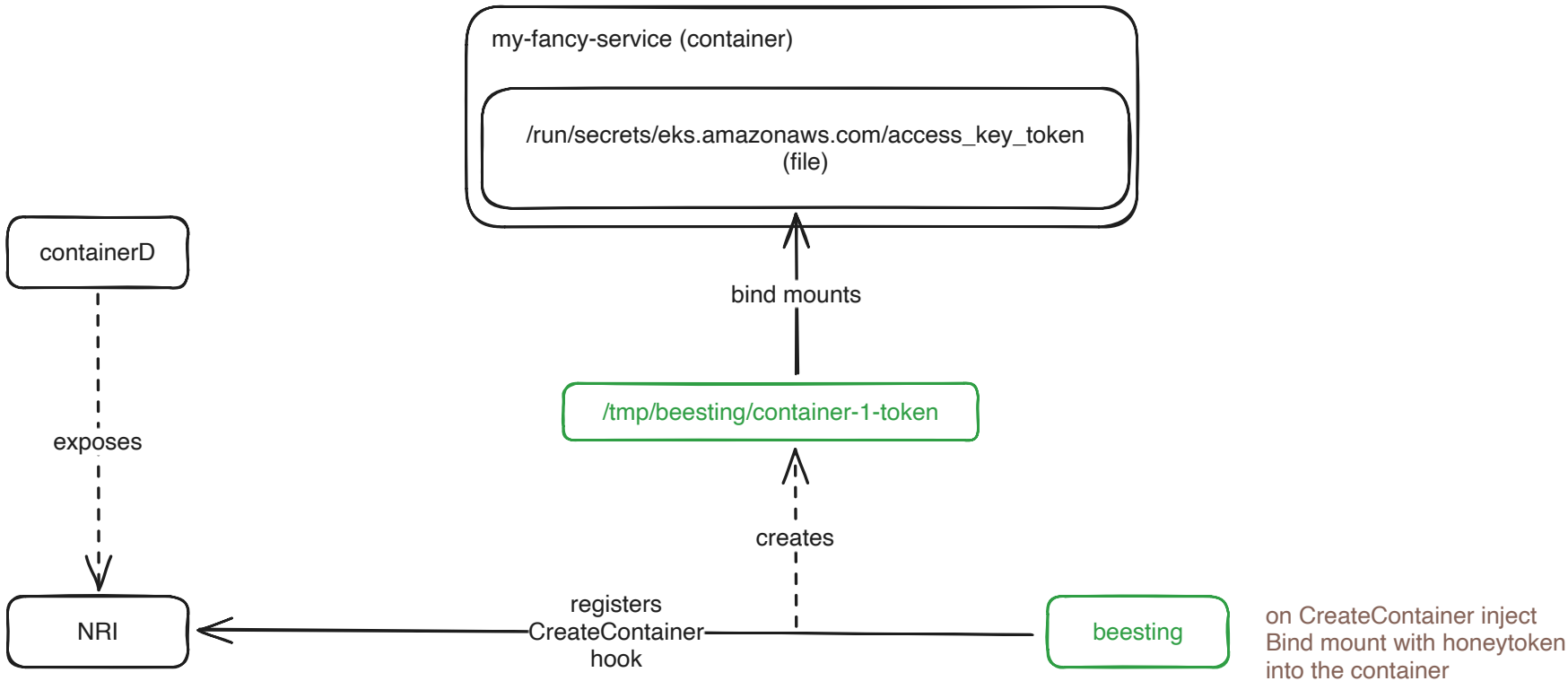
```
$ k exec deploy/dummy -it -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan  6 15:11 .
drwxr-xr-x    4 root     root        4.0K Jan  6 15:11 ..
-r--r--r--    1 root     root          16 Jan  6 15:11 access_key_token
```

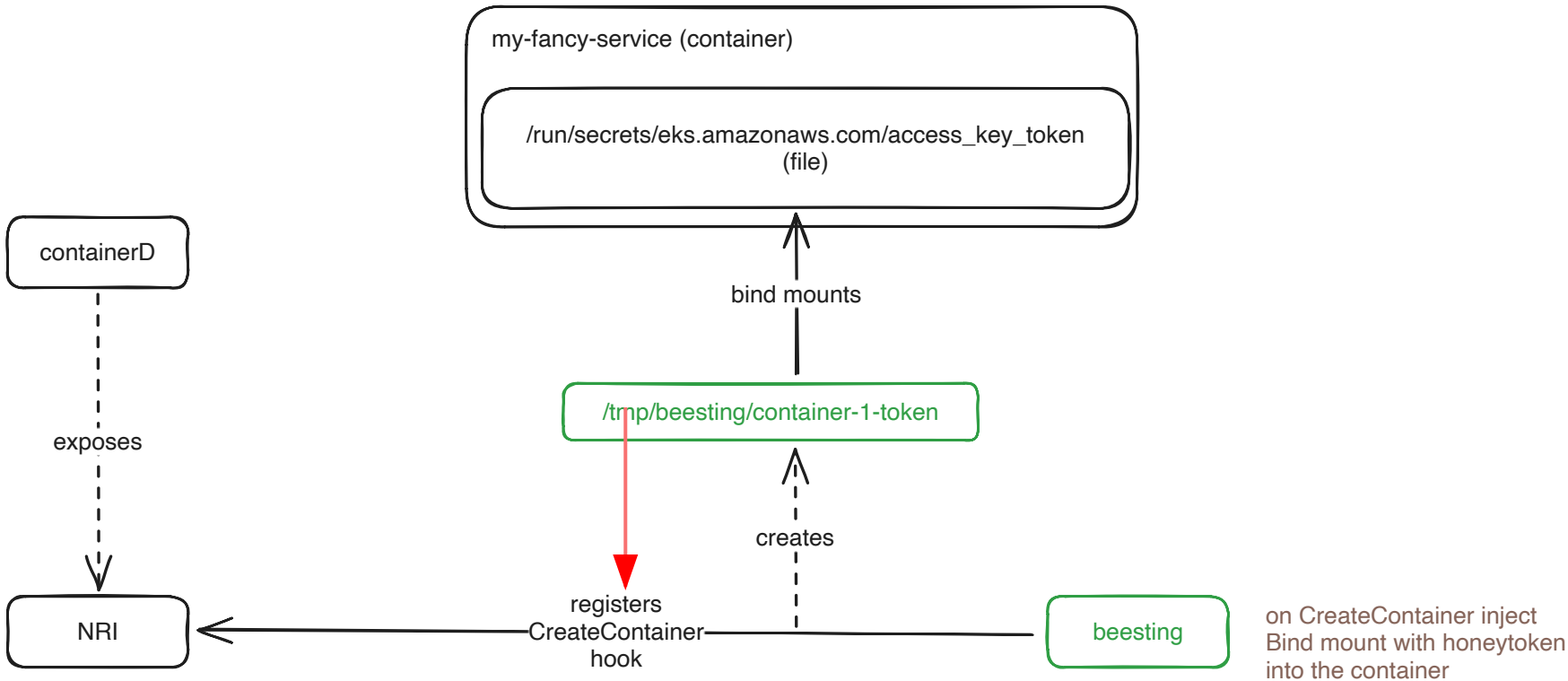
Too much complexity

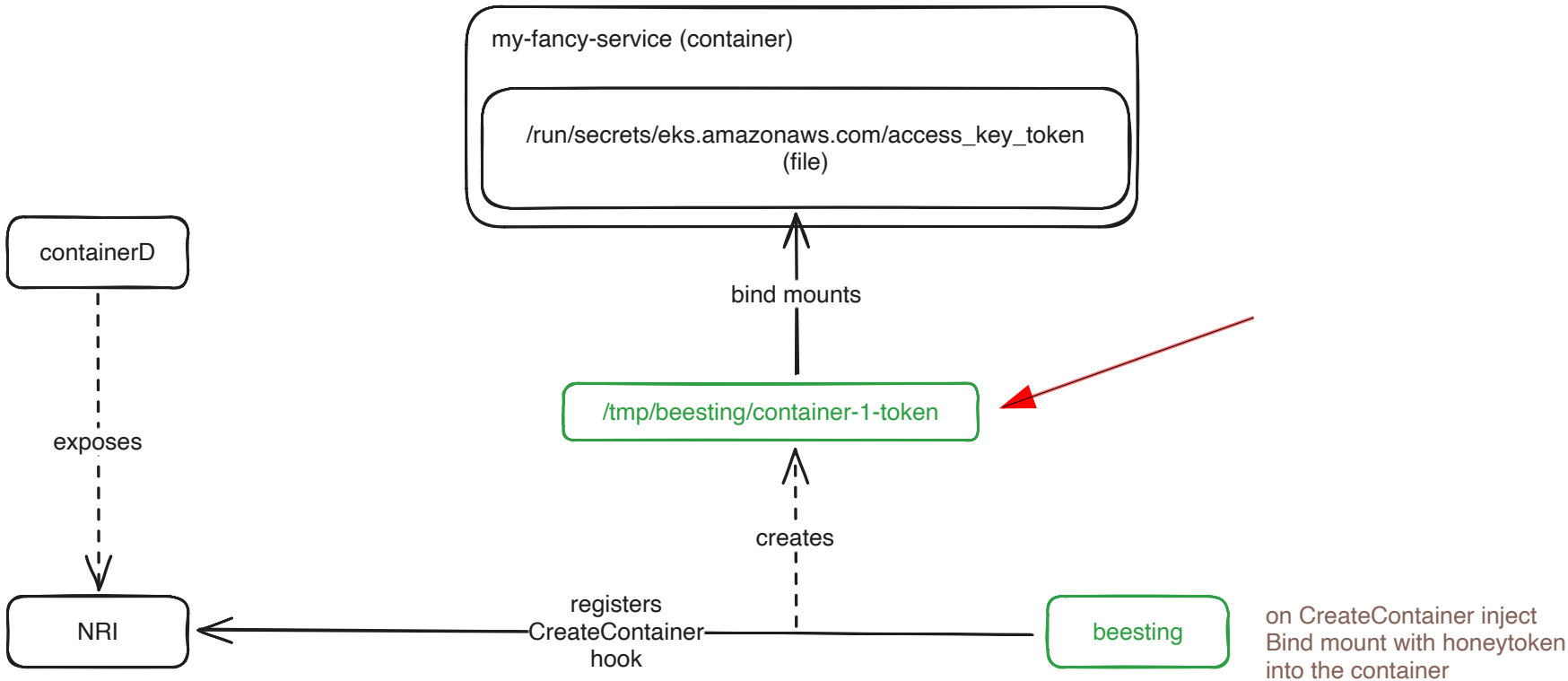
Next try

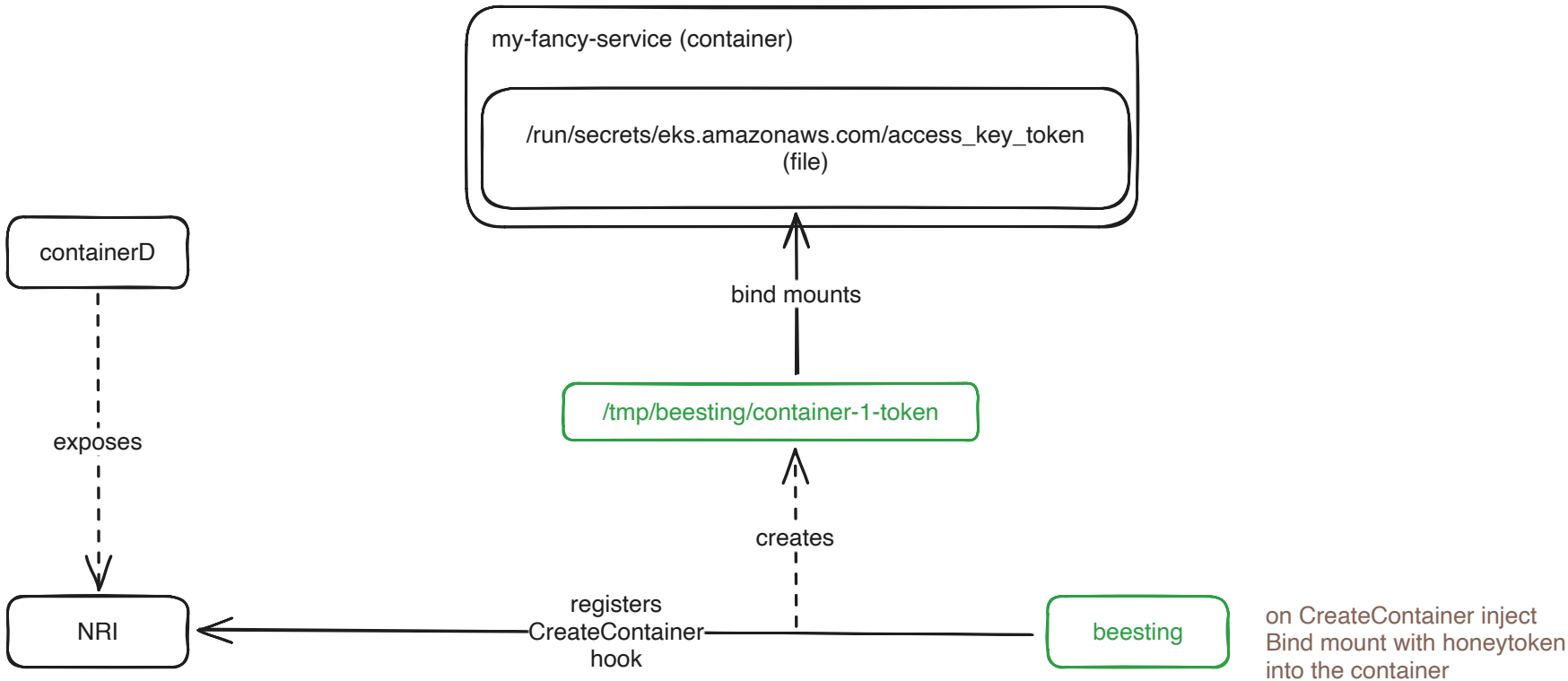
Replace hook with bind mount


host-1
file is accessible from both locations
root (mount namespace)
/tmp/beesting/container1-token
bind mount
container-1 (mount namespace)
/var/run/secrets/eks.amazon.com/access_token
/dev/sda1 (disk)
inode 123


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
/tmp/beesting/container-1-token
exposes
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
/tmp/beesting/container-1-token
exposes
creates
registers
CreateContainer
hook
NRI
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
/tmp/beesting/container-1-token
exposes
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container


my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
/tmp/beesting/container-1-token
exposes
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                        READY   STATUS              RESTARTS     AGE
beesting-agent-x2g8w        1/1     Running             0            30s
dummy-8984df79-isz1s        1/1     Running             0            52s
```

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.024708ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                        READY   STATUS              RESTARTS     AGE
beesting-agent-x2g8w        1/1     Running             0            30s
dummy-8984df79-isz1s        1/1     Running             0            52s
```

```
$ k exec deploy/dummy -it -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan  6 16:51 .
drwxr-xr-x    4 root     root        4.0K Jan  6 16:51 ..
-r--r--r--    1 root     root          16 Jan  6 16:51 access_key_token
```

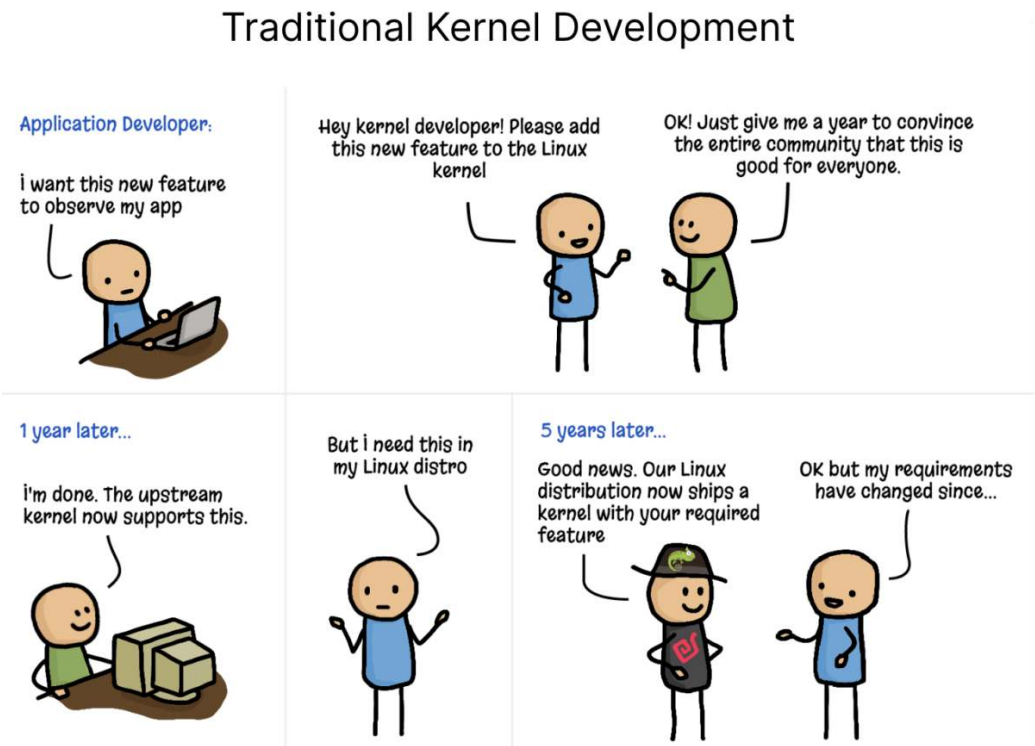
```
$ k exec deploy/dummy -it -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan  6 16:51 .
drwxr-xr-x    4 root     root        4.0K Jan  6 16:51 ..
-r--r--r--    1 root     root          16 Jan  6 16:51 access_key_token
```

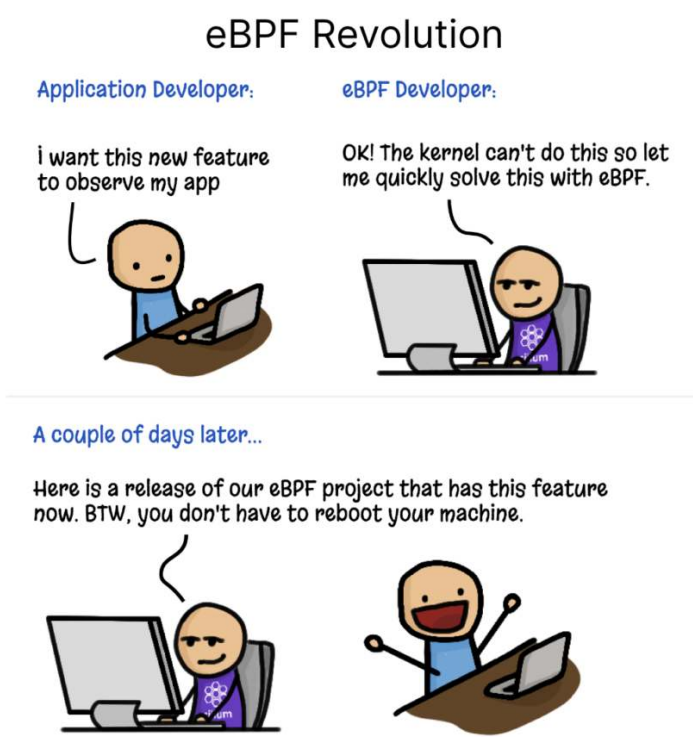
How do we detect File access?

Image by benzoix on Freepik

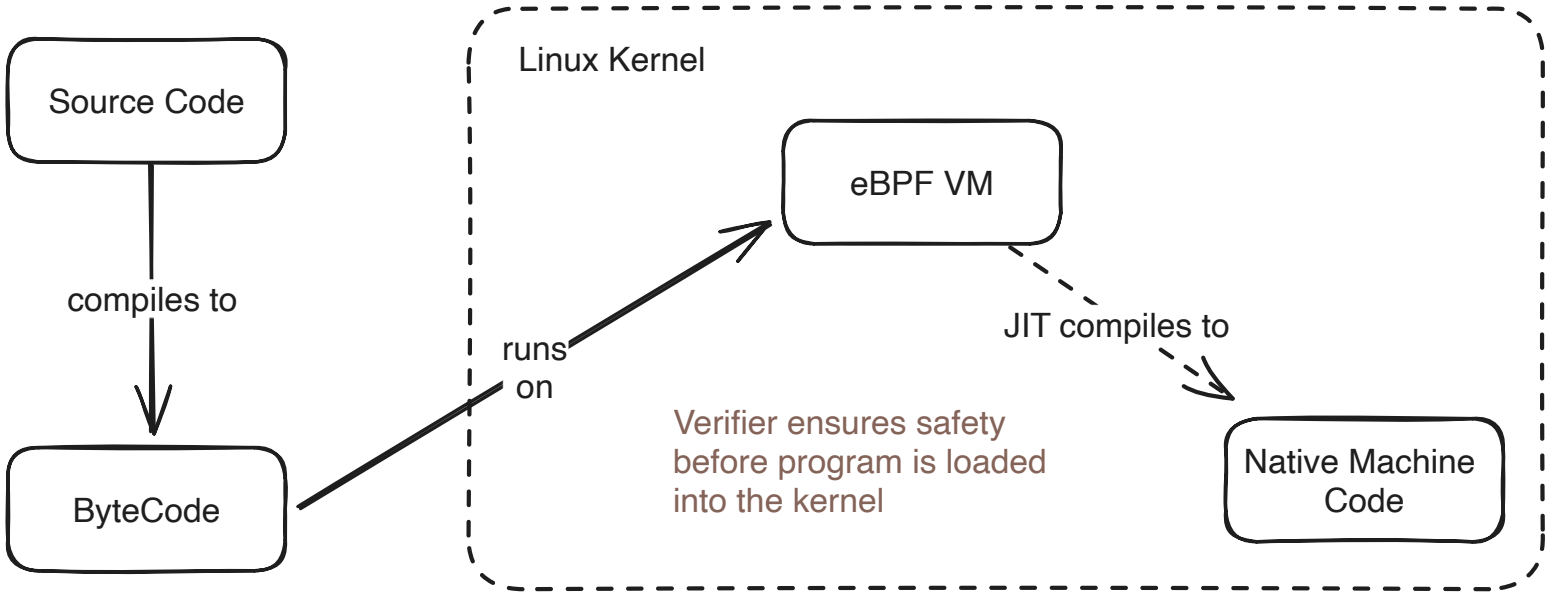

EBPF
SO HOT RIGHT NOW


eBPF





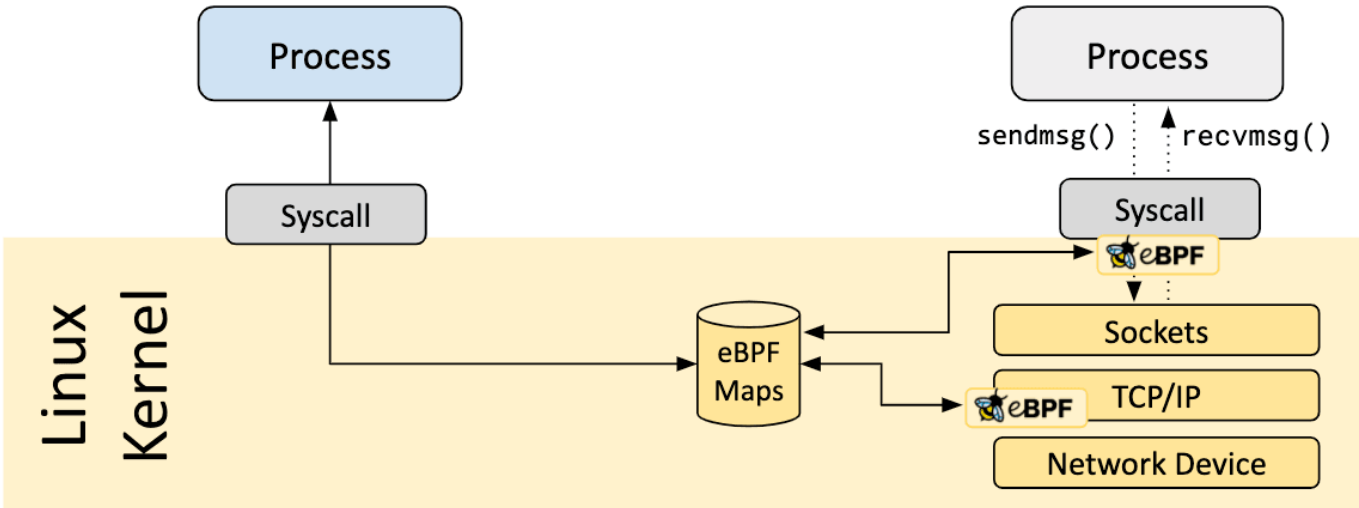
eBPF Comic by Philipp Meier and Thomas Graf


Source Code
compiles to
ByteCode
runs on
Linux Kernel
eBPF VM
JIT compiles to
Verifier ensures safety before program is loaded into the kernel
Native Machine Code

Unbounded Loops

Unbounded Loops

Image by Freepik on Freepik


Process
Process
sendmsg()
recvmsg()
Syscall
Syscall
eBPF
Linux Kernel
eBPF Maps
Sockets
eBPF
TCP/IP
Network Device

Map Types

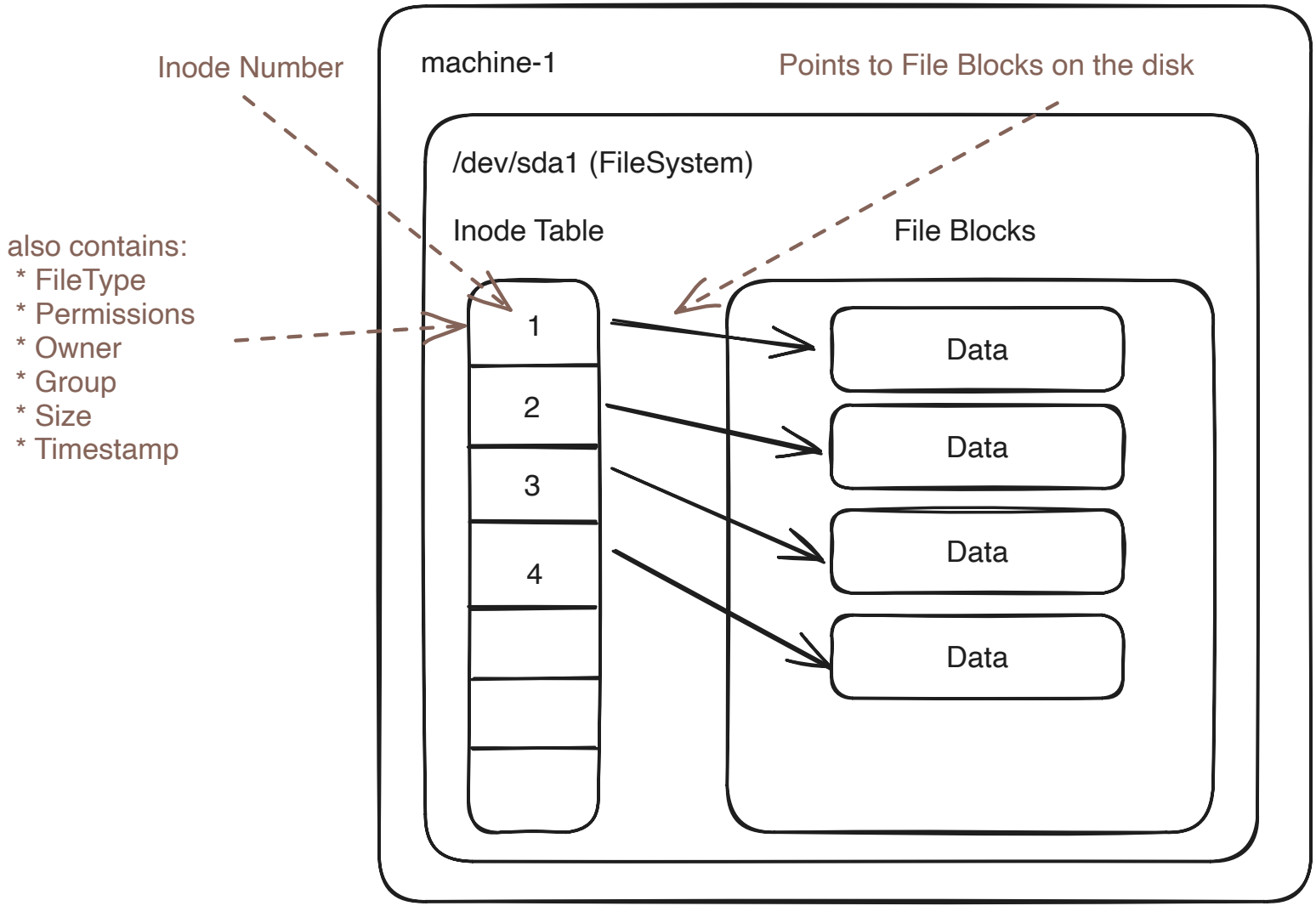
- HashTable, Arrays
- LRU (Least Recently Used)
- Perf and Ring Buffer
- ...


Linux Kernel
[...]
num = bpf_get_prandom_u32();
[...]
Process
sendmsg()
recvmsg()
Syscall
eBPF
Sockets
eBPF
TCP/IP
Network Device

Helpers

- bpf_get_current_pid_tgid
- bpf_map_lookup_elem
- bpf_map_delete_elem
- ...

What is a file?

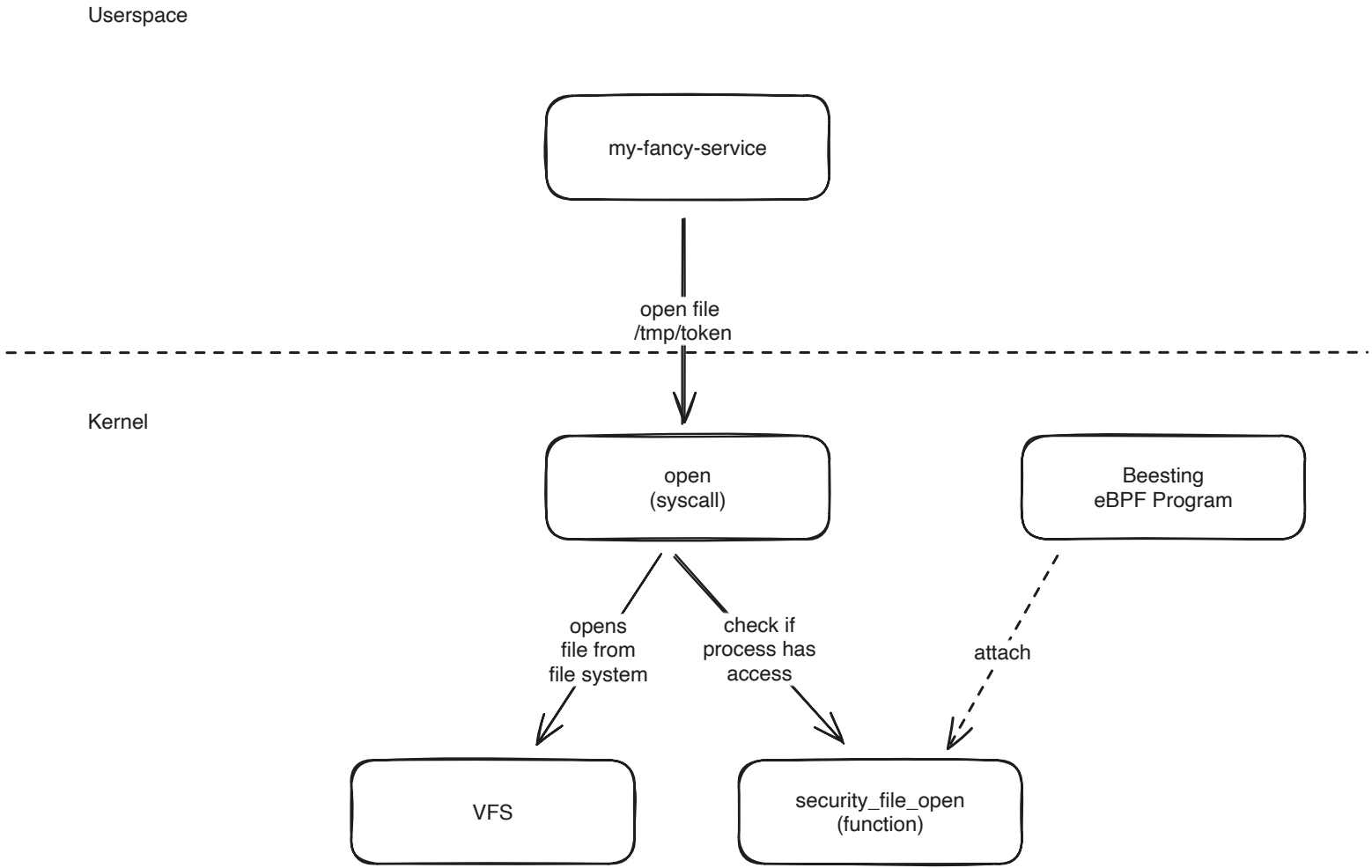

Inode Number
machine-1
Points to File Blocks on the disk
/dev/sda1 (FileSystem)
Inode Table
File Blocks
also contains:
* FileType
* Permissions
* Owner
* Group
* Size
* Timestamp
1
2
3
4
Data
Data
Data
Data

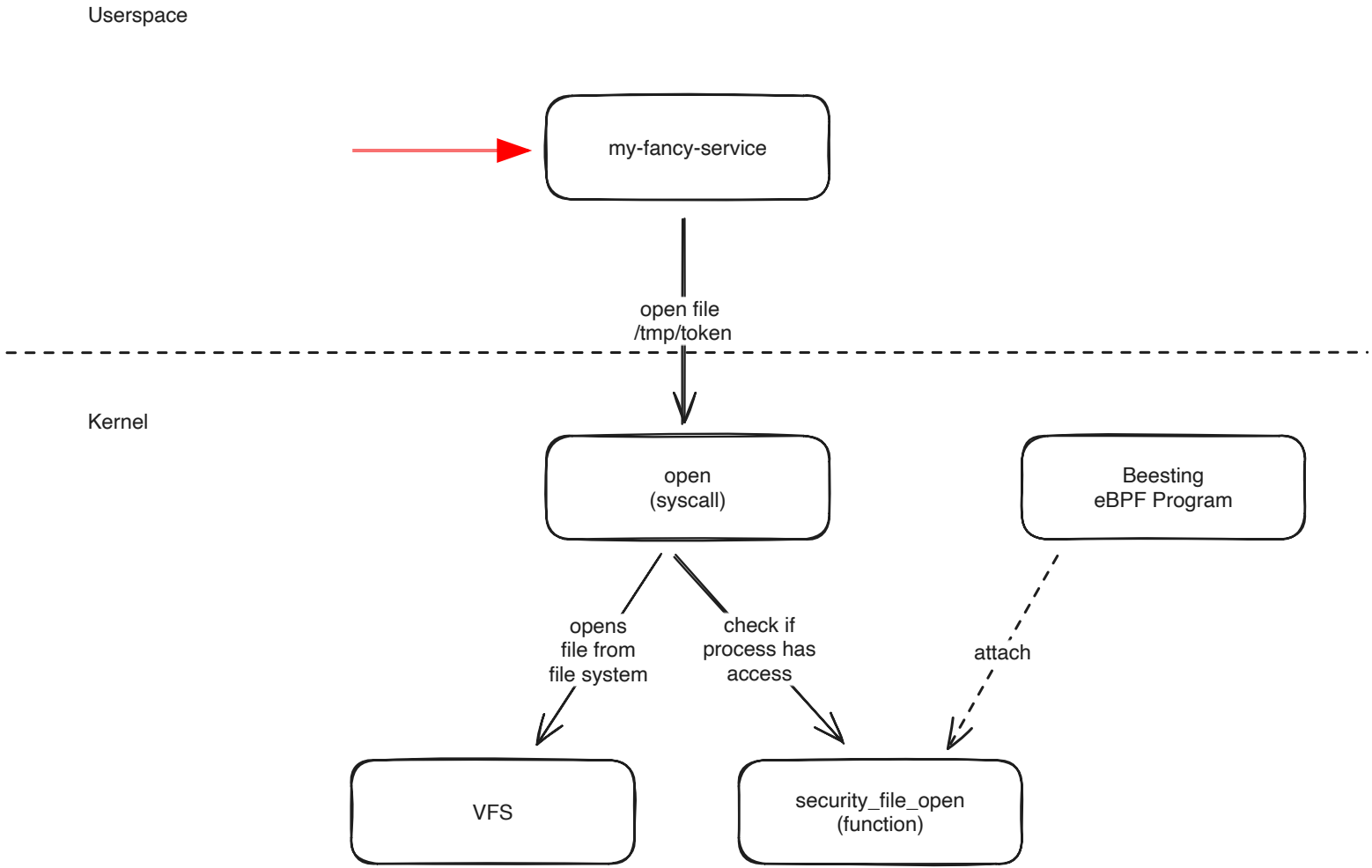
- open
- openat
- symlinks
- ...

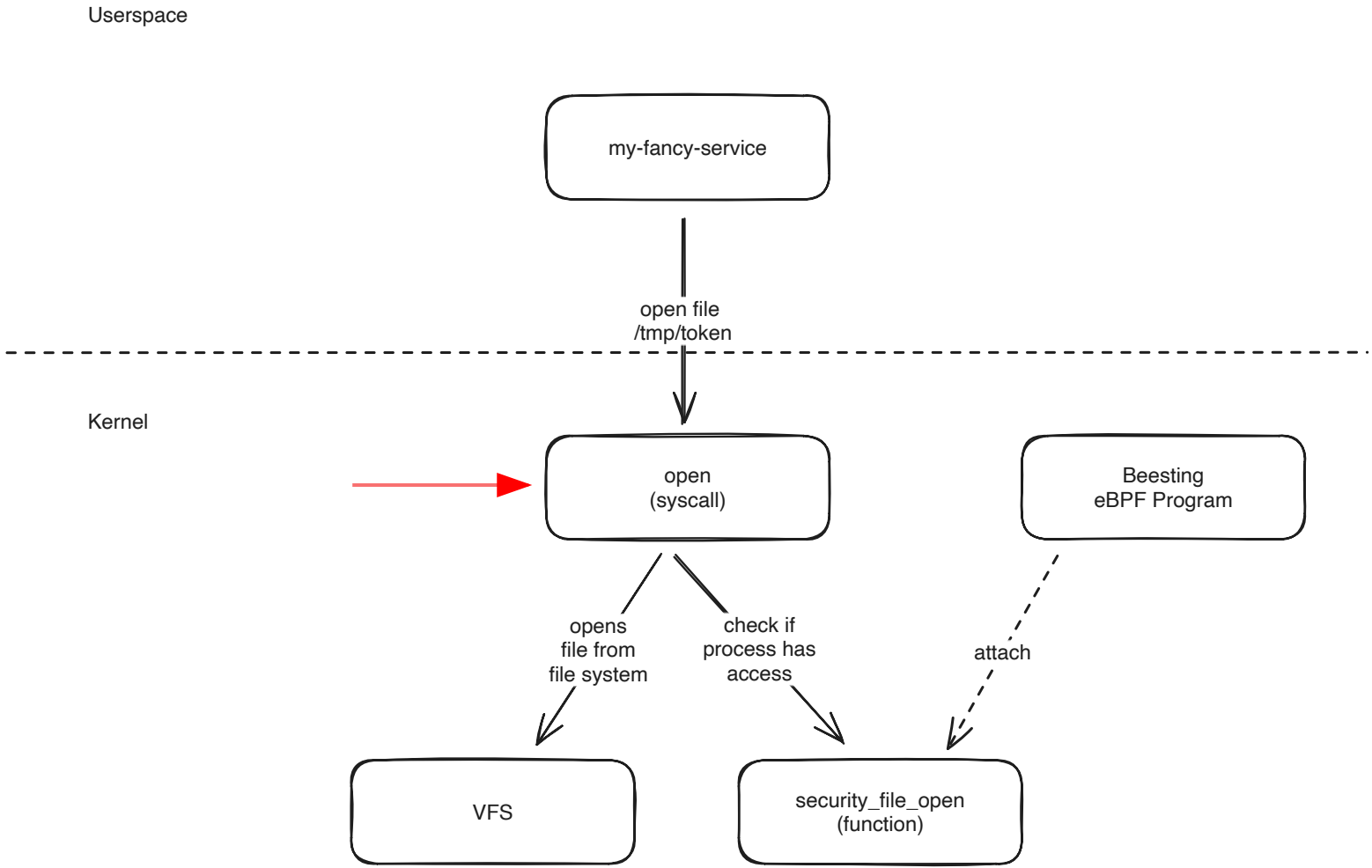
Image by wayhomestudio on Freepik

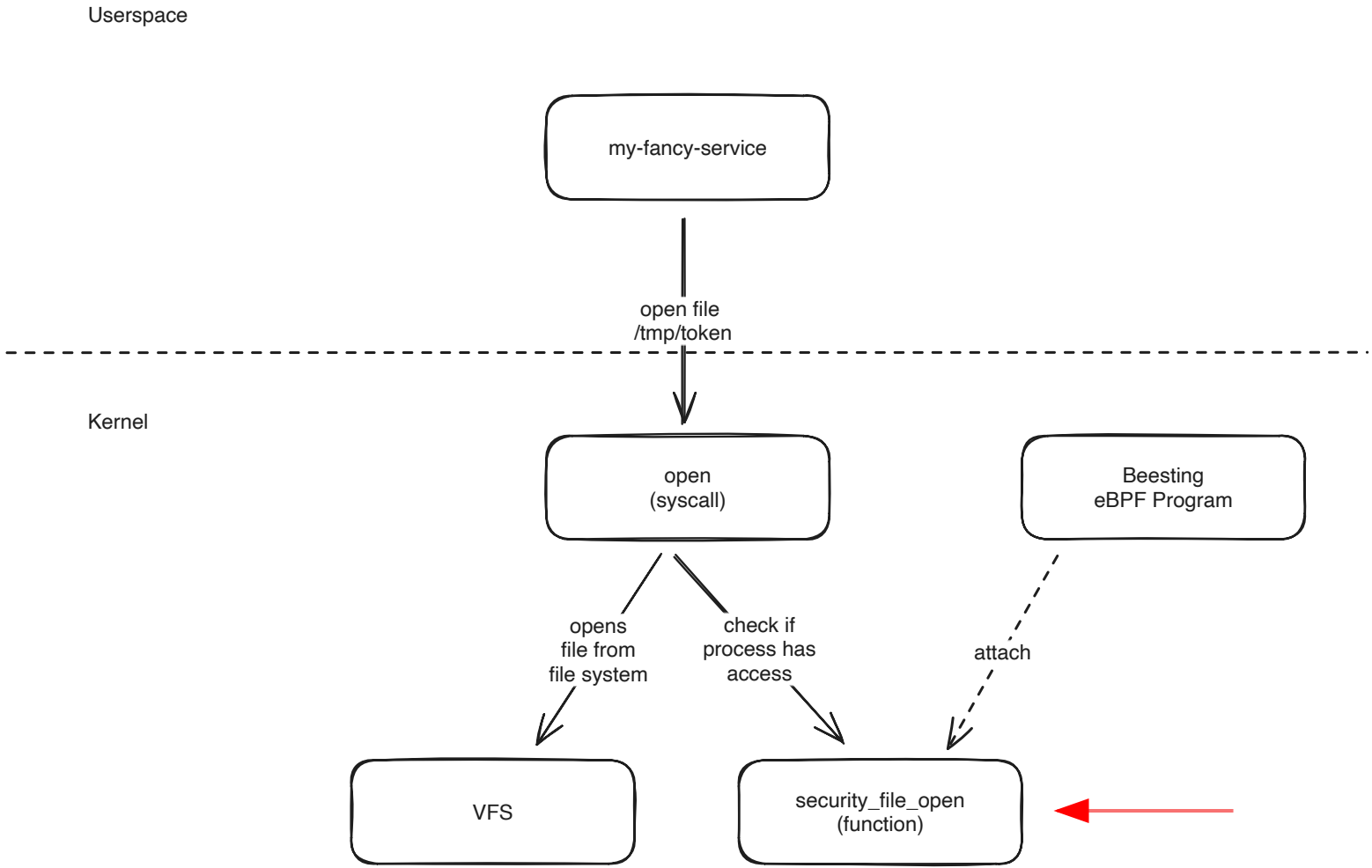
Linux Security Modules

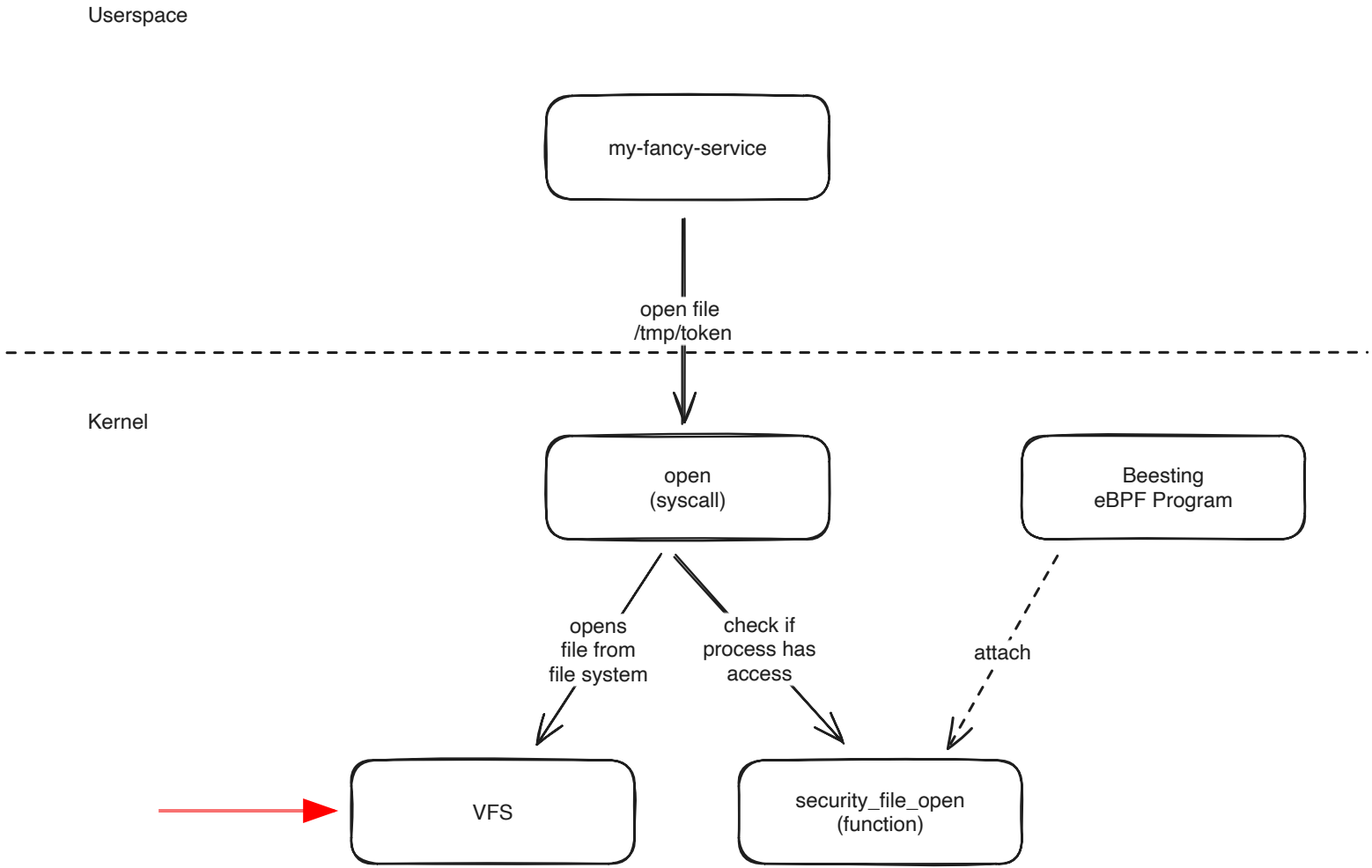
Beesting doesn't use LSM directly

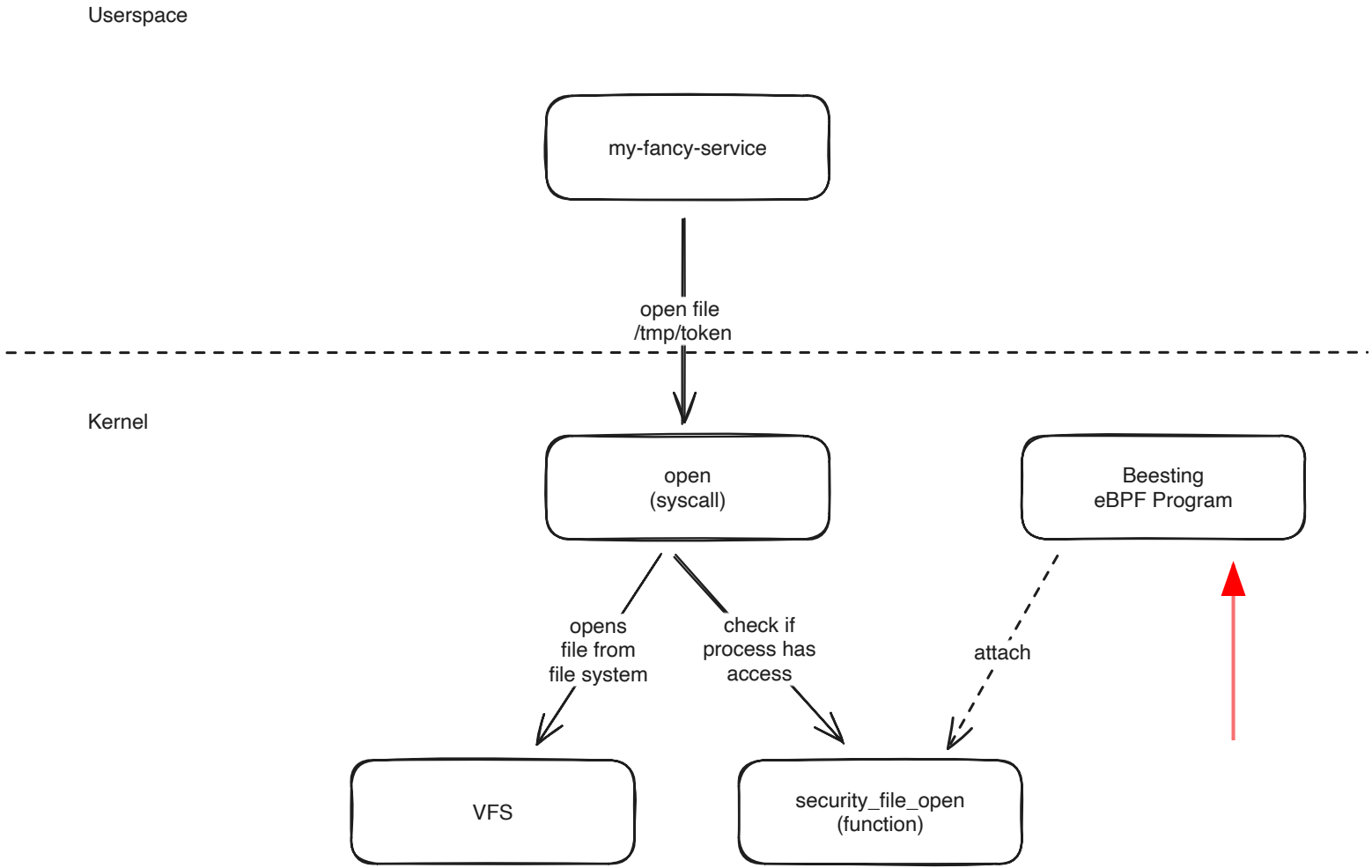

Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)


Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)

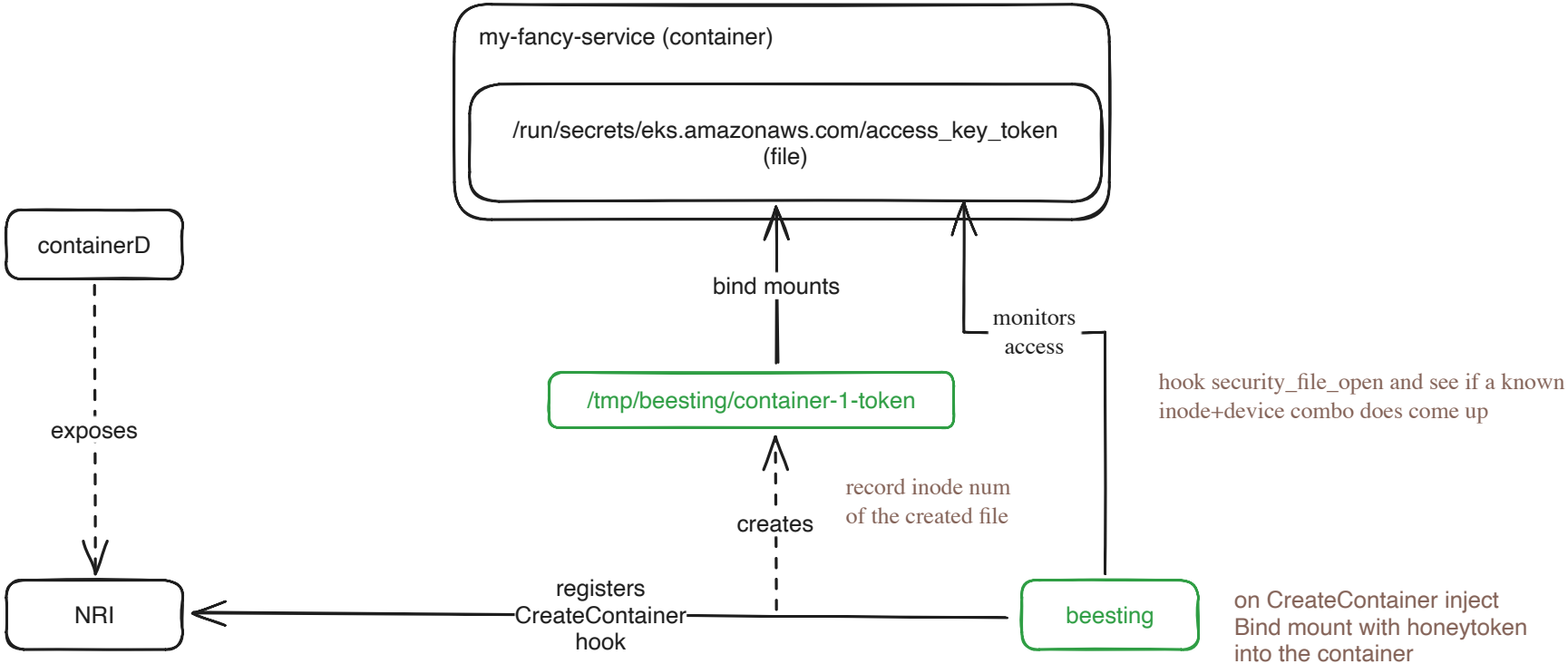

Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)

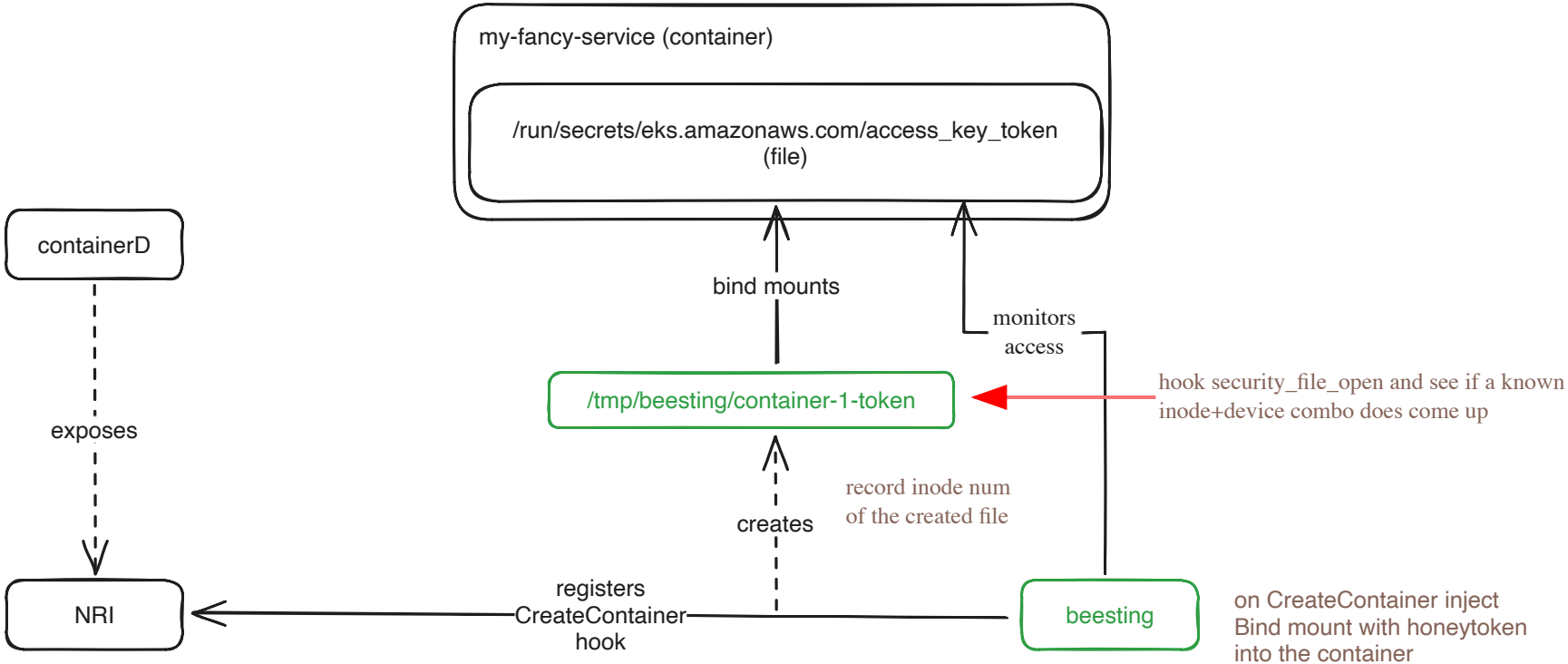

Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)

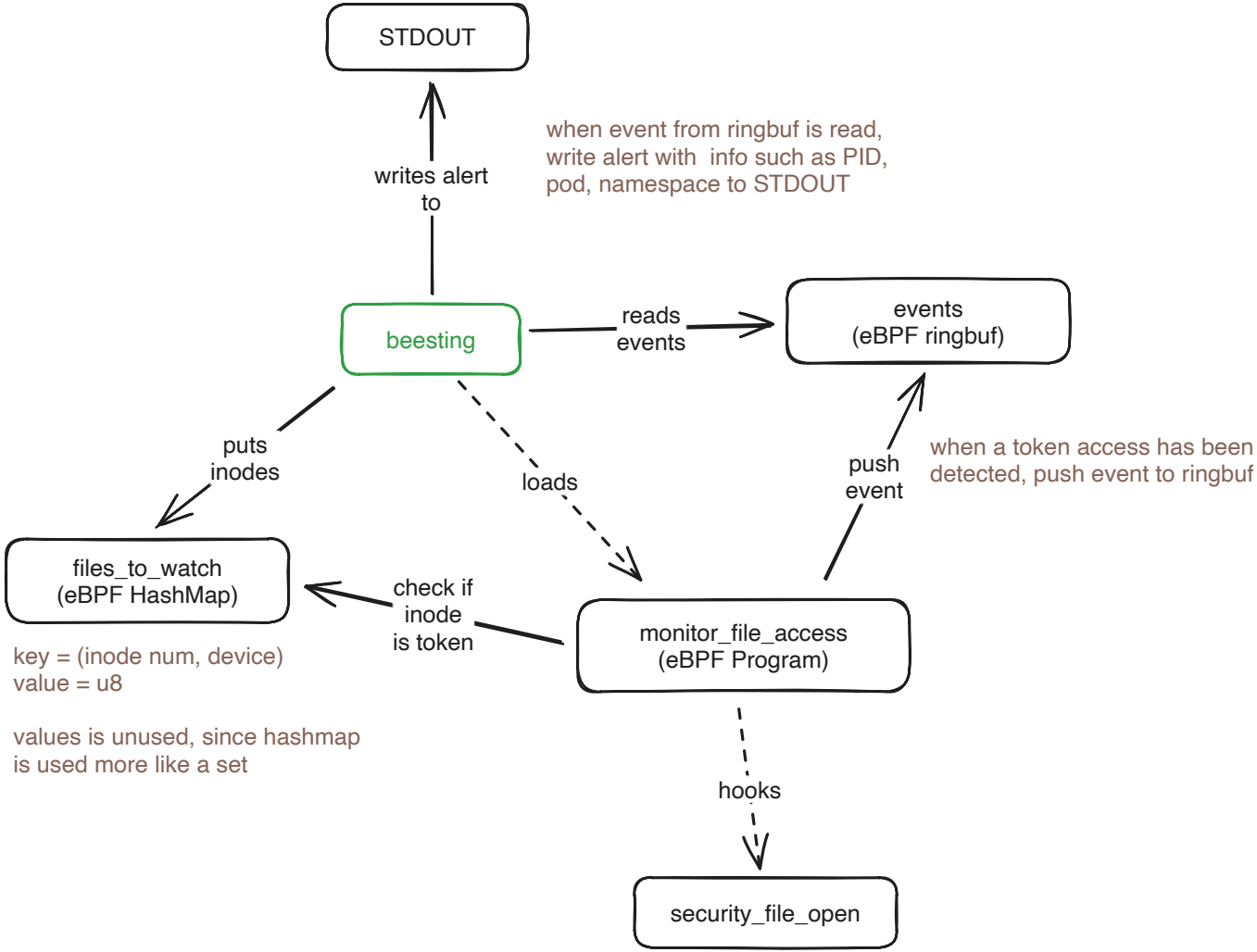

Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)

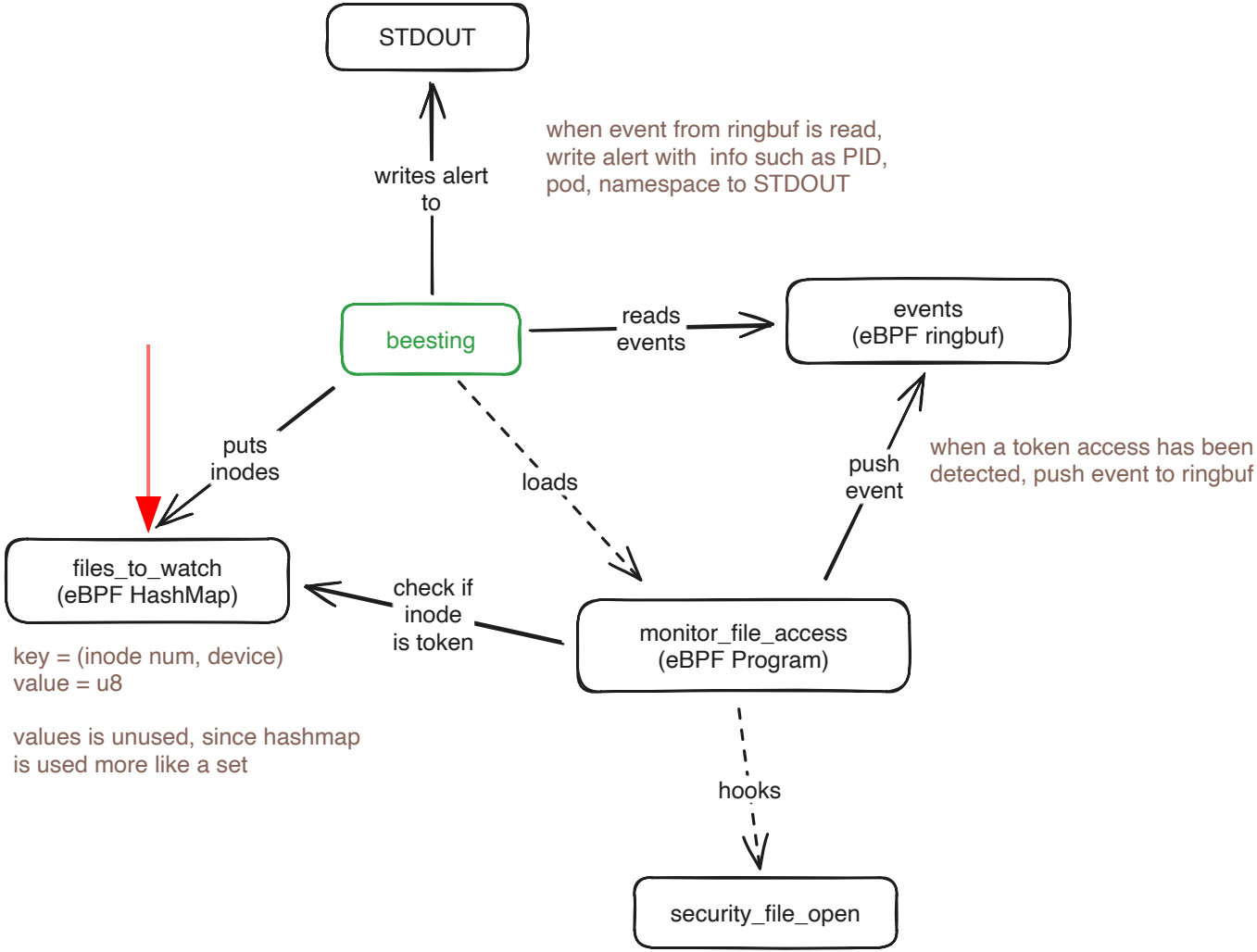

Userspace
my-fancy-service
open file
/tmp/token
Kernel
open
(syscall)
Beesting
eBPF Program
opens
file from
file system
check if
process has
access
attach
VFS
security_file_open
(function)

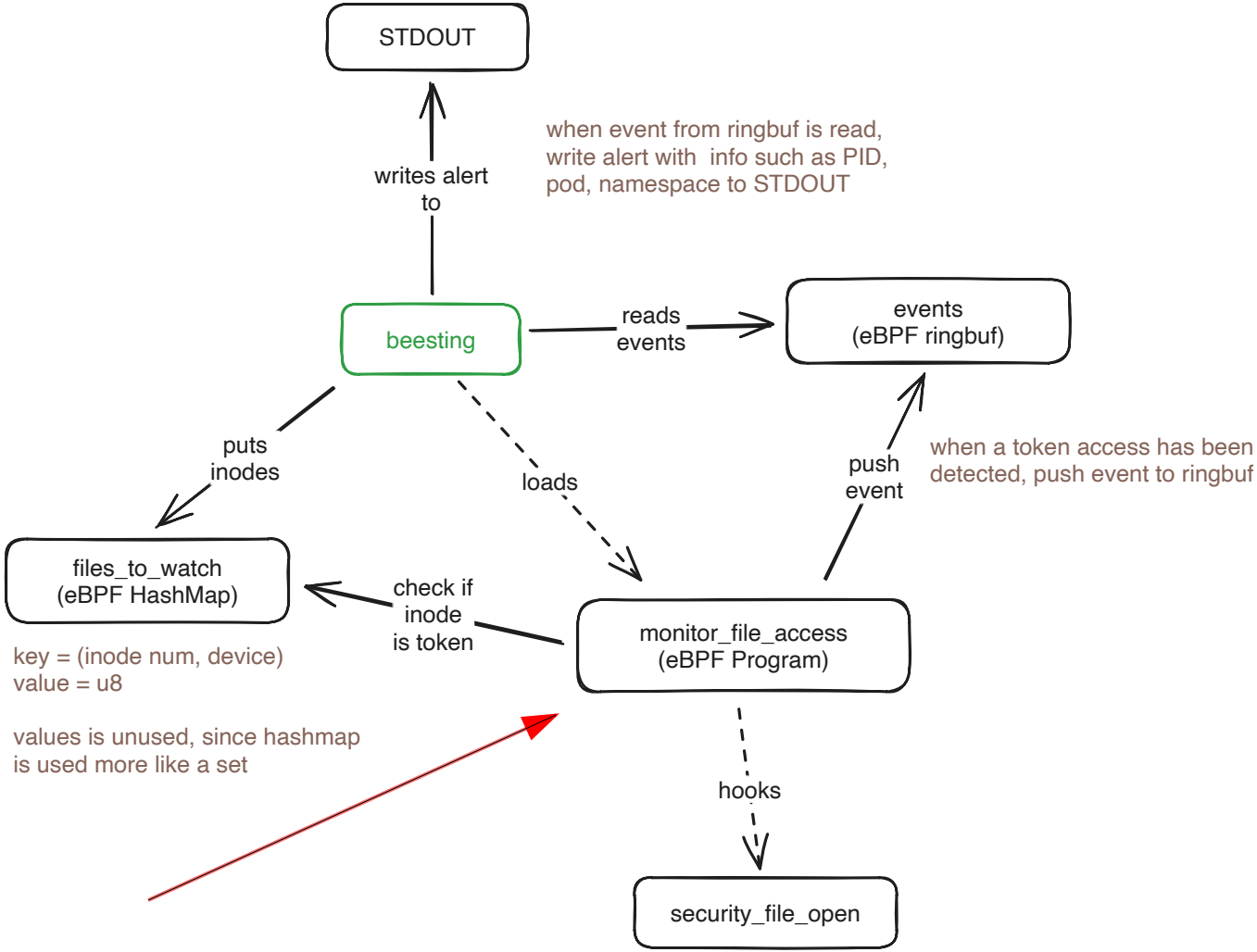
PoCv3 is based on PoCv2

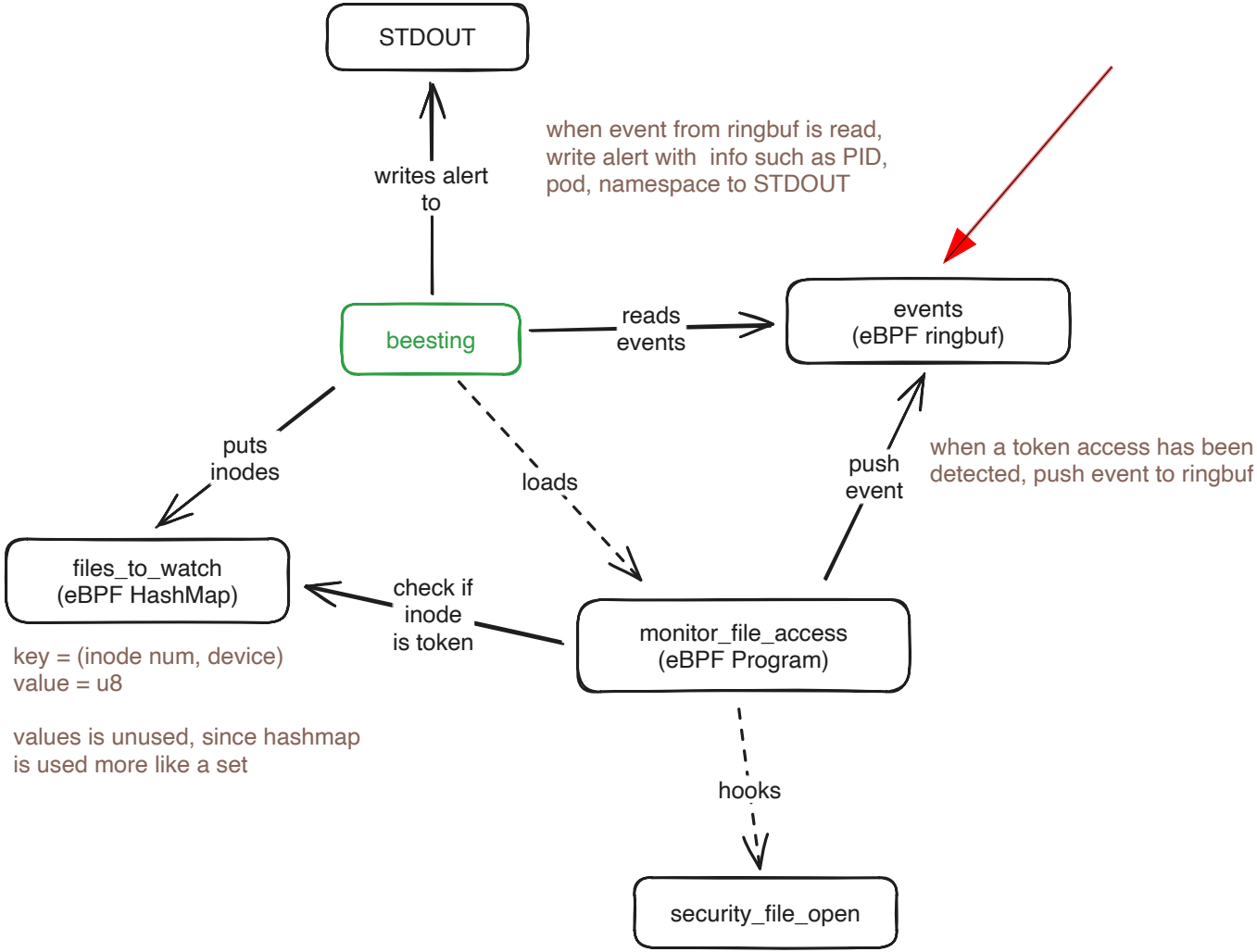

my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
monitors
access
hook security_file_open and see if a known
inode+device combo does come up
/tmp/beesting/container-1-token
exposes
record inode num
of the created file
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container

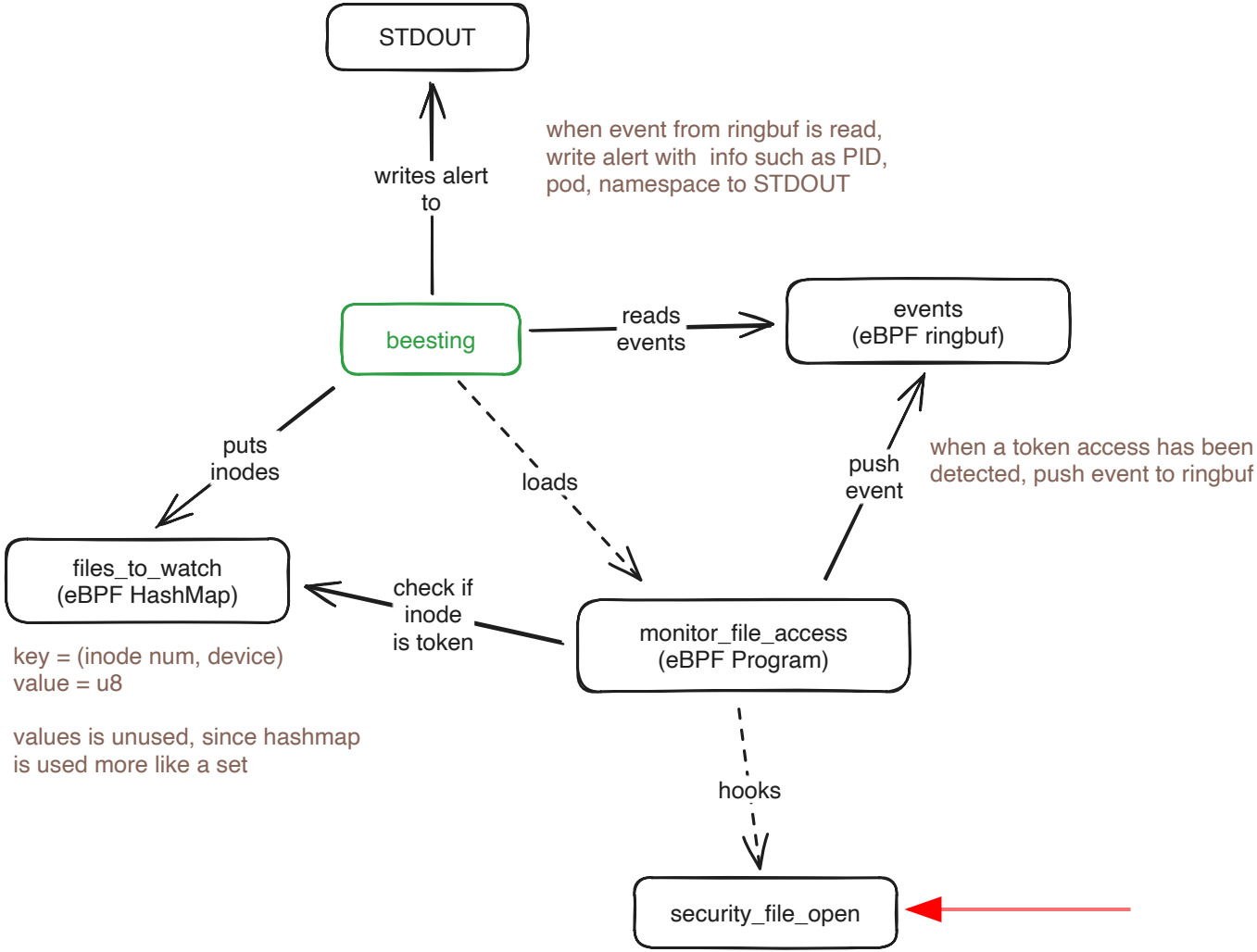

my-fancy-service (container)
/run/secrets/eks.amazonaws.com/access_key_token
(file)
containerD
bind mounts
monitors
access
/tmp/beesting/container-1-token
hook security_file_open and see if a known
inode+device combo does come up
exposes
record inode num
of the created file
creates
NRI
registers
CreateContainer
hook
beesting
on CreateContainer inject
Bind mount with honeytoken
into the container


STDOUT
writes alert to
when event from ringbuf is read, write alert with info such as PID, pod, namespace to STDOUT
beesting
reads events
events (eBPF ringbuf)
puts inodes
loads
push event
when a token access has been detected, push event to ringbuf
files_to_watch (eBPF HashMap)
check if inode is token
monitor_file_access (eBPF Program)
key = (inode num, device)
value = u8
values is unused, since hashmap is used more like a set
hooks
security_file_open


STDOUT
writes alert to
when event from ringbuf is read, write alert with info such as PID, pod, namespace to STDOUT
beesting
reads events
events (eBPF ringbuf)
puts inodes
loads
push event
when a token access has been detected, push event to ringbuf
files_to_watch (eBPF HashMap)
check if inode is token
monitor_file_access (eBPF Program)
key = (inode num, device)
value = u8
values is unused, since hashmap is used more like a set
hooks
security_file_open


STDOUT
writes alert to
when event from ringbuf is read, write alert with info such as PID, pod, namespace to STDOUT
beesting
reads events
events (eBPF ringbuf)
puts inodes
loads
push event
when a token access has been detected, push event to ringbuf
files_to_watch (eBPF HashMap)
check if inode is token
monitor_file_access (eBPF Program)
key = (inode num, device)
value = u8
values is unused, since hashmap is used more like a set
hooks
security_file_open


STDOUT
writes alert to
when event from ringbuf is read, write alert with info such as PID, pod, namespace to STDOUT
beesting
reads events
events (eBPF ringbuf)
puts inodes
loads
push event
when a token access has been detected, push event to ringbuf
files_to_watch (eBPF HashMap)
check if inode is token
monitor_file_access (eBPF Program)
key = (inode num, device)
value = u8
values is unused, since hashmap is used more like a set
hooks
security_file_open


STDOUT
writes alert to
when event from ringbuf is read, write alert with info such as PID, pod, namespace to STDOUT
beesting
reads events
events (eBPF ringbuf)
puts inodes
loads
push event
when a token access has been detected, push event to ringbuf
files_to_watch (eBPF HashMap)
check if inode is token
monitor_file_access (eBPF Program)
key = (inode num, device)
value = u8
values is unused, since hashmap is used more like a set
hooks
security_file_open

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.103810ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                    READY   STATUS    RESTARTS   AGE
beesting-agent-svpzl    1/1     Running   0          20s
dummy-8984df79-kddjh    1/1     Running   0          13s
```

```
$ skaffold run
...
Waiting for deployments to stabilize...
Deployments stabilized in 5.103810ms
You can also run [skaffold run --tail] to get the logs

$ k delete -f HACK/dummy.yaml
deployment.apps/dummy deleted

$ k apply -f HACK/dummy.yaml
deployment.apps/dummy created

$ k get pods
NAME                    READY   STATUS    RESTARTS   AGE
beesting-agent-svpzl    1/1     Running   0          20s
dummy-8984df79-kddjh    1/1     Running   0          13s
```

```
$ k exec deploy/dummy -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan 11 09:14 .
drwxr-xr-x    4 root     root        4.0K Jan 11 09:14 ..
-rw-r--r--    1 root     root          16 Jan 11 09:14 access_key_token
```

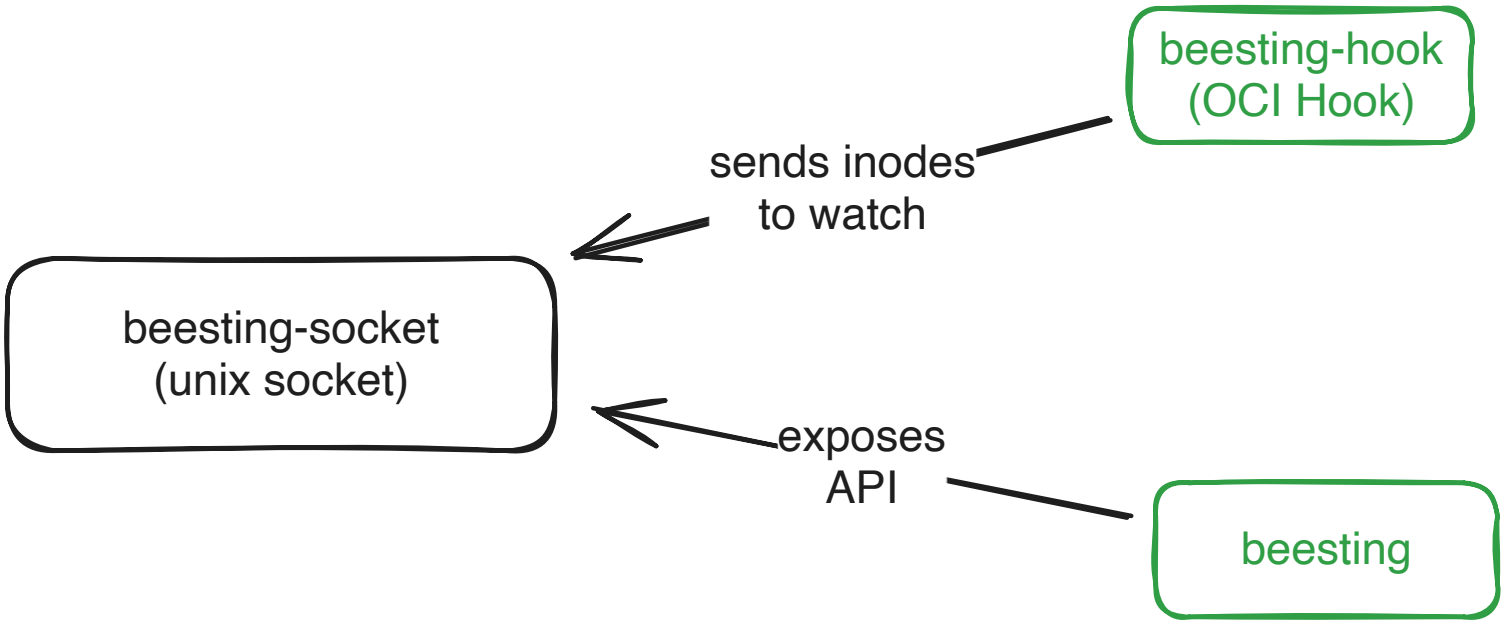
```
$ k exec deploy/dummy -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan 11 09:14 .
drwxr-xr-x    4 root     root        4.0K Jan 11 09:14 ..
-rw-r--r--    1 root     root          16 Jan 11 09:14 access_key_token
```

```
$ k exec deploy/dummy -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan 11 09:14 .
drwxr-xr-x    4 root     root        4.0K Jan 11 09:14 ..
-rw-r--r--    1 root     root          16 Jan 11 09:14 access_key_token

$ k exec deploy/dummy -- cat /var/run/secrets/eks.amazonaws.com/access_key_token
2yWeFNuHzb7wUw==
```

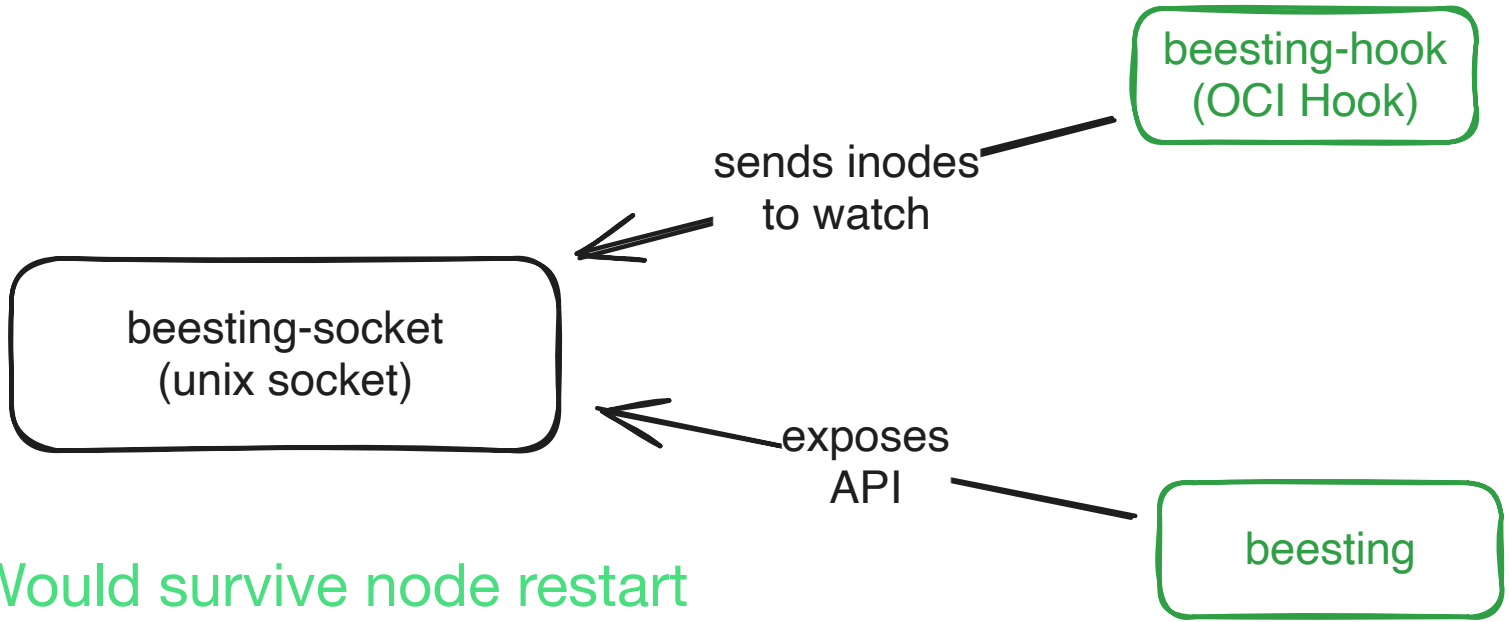
```
$ k exec deploy/dummy -- ls -alh /var/run/secrets/eks.amazonaws.com/
total 12K
drwxr-xr-x    2 root     root        4.0K Jan 11 09:14 .
drwxr-xr-x    4 root     root        4.0K Jan 11 09:14 ..
-rw-r--r--    1 root     root          16 Jan 11 09:14 access_key_token

$ k exec deploy/dummy -- cat /var/run/secrets/eks.amazonaws.com/access_key_token
2yWeFNuHzb7wUw==

$ k logs ds/beesting-agent --tail=2
time=2025-01-11T09:14:37.455Z level=DEBUG msg="watch inode" token.Inode=92 token.Dev=78
2025-01-11T09:23:17Z 💥 Honey Token Access Detected!
    Pod: default/dummy-8984df79-kddjh
    Container: dummy-pod,
    PID: 379512,
    StartTime: 584352740276296
    Comm: cat
```

Further research

File injection based on PoCv1



monitoring is the same as PoCv3



Would survive node restart

How to secure it?

monitoring is the same as PoCv3

Further research

Hook read operations and scan for pattern

Very flexible

https://github.com/patrickpichler/beesting/

https://patrickpichler.dev

Home

Posts

Figuring out which helpers are available in what kernel version in eBPF

eBPF helpers are a vital part of any eBPF program. It is often not easy to figure out, which helper you have available for a certain program type at a given Linux Kernel Version. The goal of this blog post is,...

November 10, 2024 · 6 min · Patrick Pichler

Hello Blog

Hello world from patrickpichler.dev I am planning to write about different kubernetes/cloud/container related topics. So stay tuned!

September 26, 2023 · 1 min · Patrick Pichler