
The Hitchhiker's Guide to Pod Security
@LachlanEvenson


My goal for this session?
…everyone here
enables Pod Security
on the clusters they manage!
@LachlanEvenson

What you'll leave with today

- Pod Security Concepts 🐟
- Pod Security in Action 🤖
- Next Steps 🐬

About me

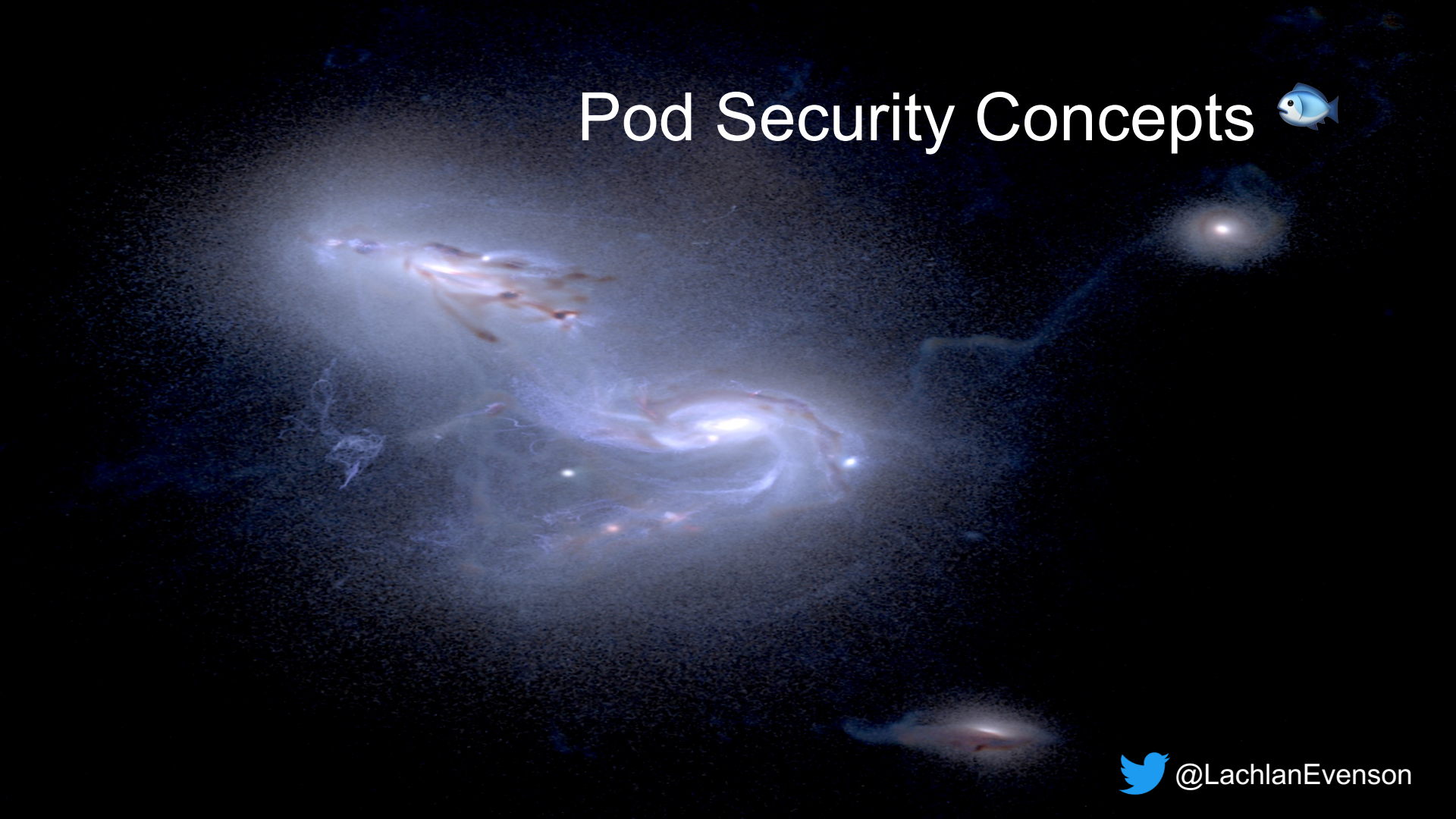
Hi! I'm Lachlan (Lachie) Evenson

- Product Manager @ Azure Upstream
- CNCF Ambassador
- CNCF Governing Board member
- Emeritus Kubernetes Steering Committee
- Emeritus Kubernetes 1.16 release lead

I love

- Travel
- Language
- Hiking
- Experiencing different cultures

@LachlanEvenson


Pod Security Concepts
@LachlanEvenson

What is Pod Security?

- Pod Security is a **built-in** admission controller
- Evaluates **pod specifications** against a predefined set of **Pod Security Standards** (we will cover this later)
- Applied at the **namespace** level when pods are **created**
- **Beta** as of Kubernetes v1.23
- **Planned** for stable in v1.25

https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/

Why Pod Security?

- Provides policy standards to **restrict** Pod privileges
- **Simple** and **easy-to-use**
- **Predefined** Pod Security levels that a cluster administrator can configure to meet the desired security posture
- **Supports** and encourages Kubernetes best practices
- Performs **validation** only (no mutation)

What about Pod Security Policy?

- PodSecurityPolicy API is deprecated and will be removed from Kubernetes in v1.25
- Pod Security Policy had a few shortcomings
- Pod Security **does not** have feature parity with Pod Security Policy
- Pod Security does not support mutation

Migration from Pod Security Policy to Pod Security

- There is a **migration path** from Pod Security Policy to Pod Security
- If you're running Pod Security Policy today and would like to know how to migrate to Pod Security
- https://kubernetes.io/docs/tasks/configure-pod-container/migrate-from-psp/
- To summarize the process:
  - Update all existing PSPs to be non-mutating
  - Eliminate options not covered by the Pod Security Standards
  - Apply Pod Security policies in warn or audit mode
  - Upgrade Pod Security policies to enforce mode
  - Remove PodSecurityPolicy from --enable-admission-plugins

Other similar ecosystem tooling

- The goal of Pod Security is to provide a built-in set of capabilities to improve the security of your workloads
- These capabilities all fall under Kubernetes policy and governance
- What if you need mutation or other more complex functionality
  - https://github.com/open-policy-agent/gatekeeper
  - https://github.com/kyverno/kyverno
- Pod Security was designed to be **composable** and work with other solutions

NB: Other options exist - these are examples to show you where to start investigating

Elements of Pod Security

- Built-in **admission controller** (may be run as a standalone admission webhook)
- There are three different policy levels known as **Pod Security Standards** that range from permissive to restrictive
- Policies are applied in a specific **mode** per namespace
- Multiple modes (with different policy levels) can be set on the same namespace.

Elements of Pod Security

- **Pod Security Standards** levels are as follows
  - `privileged` — open and unrestricted
  - `baseline` — Covers known privilege escalations while minimizing restrictions
  - `restricted` — Highly restricted, hardening against known and unknown privilege escalations. May cause compatibility issues

https://kubernetes.io/docs/concepts/security/pod-security-standards/

@LachlanEvenson

Elements of Pod Security continued

- Each of these policy levels define which fields are restricted within a pod specification and the allowed values.
  - Some of the fields restricted by these policies include:
    - `spec.securityContext.sysctls`
    - `spec.hostNetwork`
    - `spec.volumes[*].hostPath`
    - `spec.containers[*].securityContext.privileged`
- Applied via labels on Namespace resources, which allows for granular per-namespace policy selection

https://kubernetes.io/docs/concepts/security/pod-security-standards/

@LachlanEvenson

Elements of Pod Security

- Policies are applied in a specific mode
- Multiple modes (with different policy levels) can be set on the same namespace.
- List of modes:
  - `enforce` — Any Pods that violate the policy will be rejected
  - `audit` — Violations will be recorded as an annotation in the audit logs, but don't affect whether the pod is allowed.
  - `warn` — Violations will send a warning message back to the user, but don't affect whether the pod is allowed.

Elements of Pod Security continued

- In addition to modes you can also pin the policy to a specific Kubernetes version (for example v1.22).
- Pinning to a specific version allows the behavior to remain consistent if the policy definition changes in future Kubernetes releases.

Enabling Pod Security

- For Kubernetes >= v1.23.0 it will be enabled by default
- For Kubernetes == v1.22.0 you will need to set the PodSecurity=true feature gate on kube-apiserver
- For Kubernetes version > v1.22.0 that support admission webhooks there is separate documentation on the install https://github.com/kubernetes/pod-security-admission/tree/master/webhook

NB. Please don't test this in Production! Get familiar with Pod Security in a development or lab cluster

Configuring Pod Security

- Policies are applied to a namespace via labels
- These labels are as follows:
  - `pod-security.kubernetes.io/<MODE>: <LEVEL>` (required to enable pod security)
  - `pod-security.kubernetes.io/<MODE>-version: <VERSION>` (optional, defaults to latest)
- <OPTIONAL> cluster-wide policies and exemptions using the AdmissionConfiguration*.
- The possible <MODE(S)> are **enforce**, **audit** and **warn**.

* AdmissionConfiguration must be configured via apiserver flags

Configuring Pod Security continued

- A specific version can be supplied for each enforcement mode.
- The typical uses for **warn** are to get ready for a future change where you want to enforce a different policy.
- **enforce** is only applied to pod resource. **warn** and **audit** are also applied to workload resources

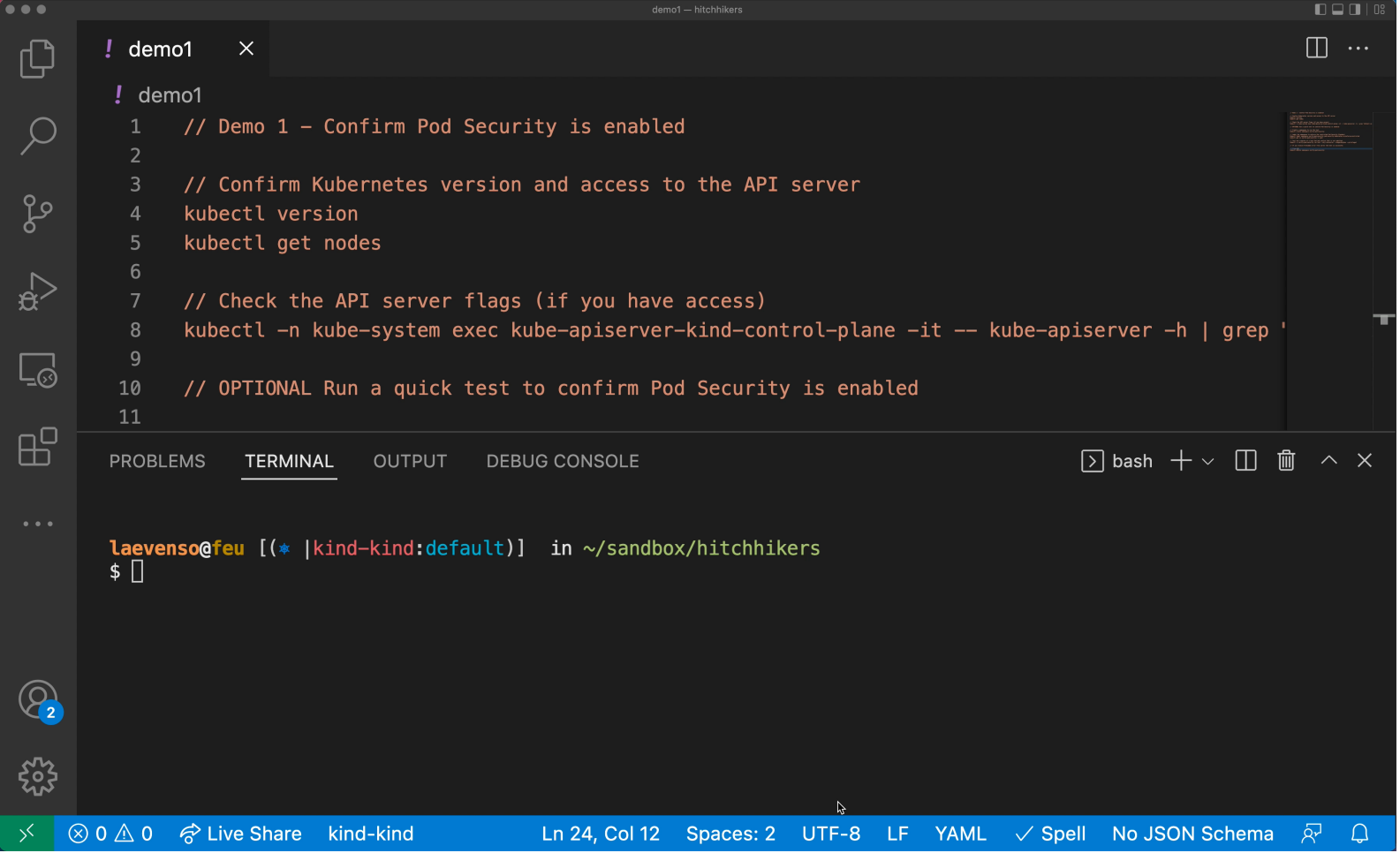
Configuring Pod Security continued

- The audit mode logs to the cluster audit log (example audit log below)


```
{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"","pod-security.kubernetes.io/audit":"allowPrivilegeEscalation != false (container \"busybox\" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container \"busybox\" must set securityContext.capabilities.drop=[\"ALL\"]), runAsNonRoot != true (pod or container \"busybox\" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container \"busybox\" must set securityContext.seccompProfile.type to \"RuntimeDefault\" or \"Localhost\")"}}
```

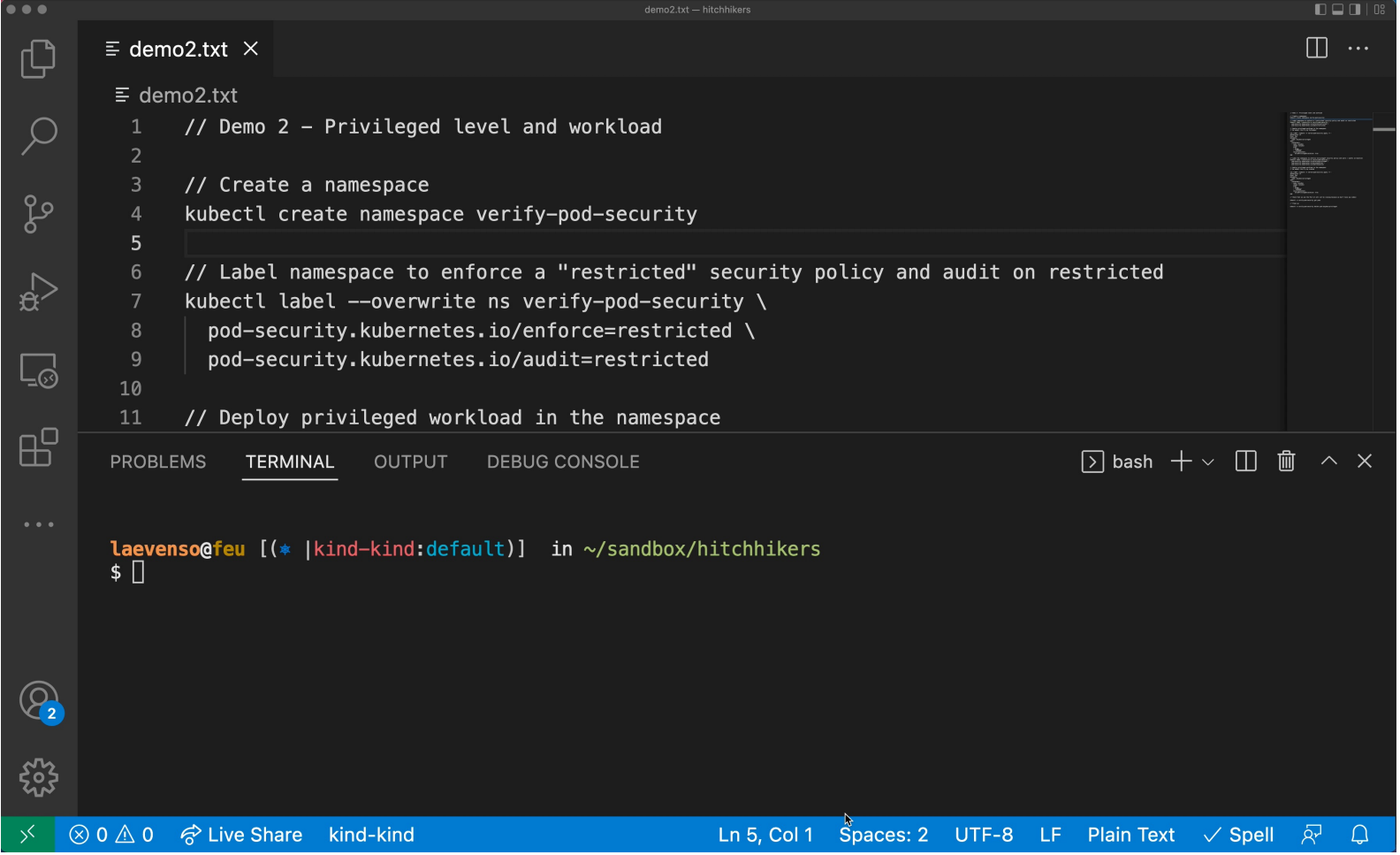
Pod Security in Action 🤖

@LachlanEvenson



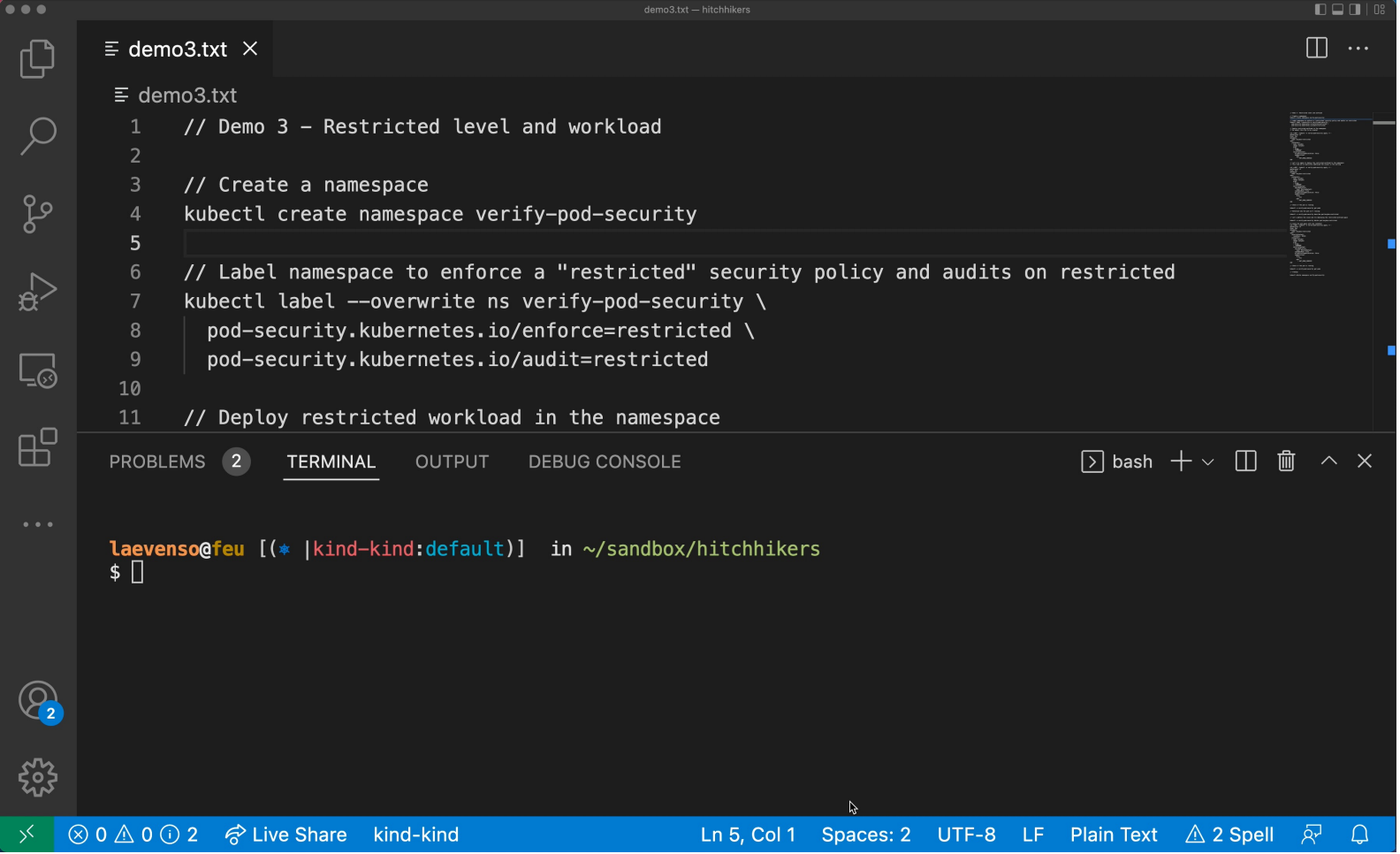
Confirm Pod Security is enabled

@LachlanEvenson



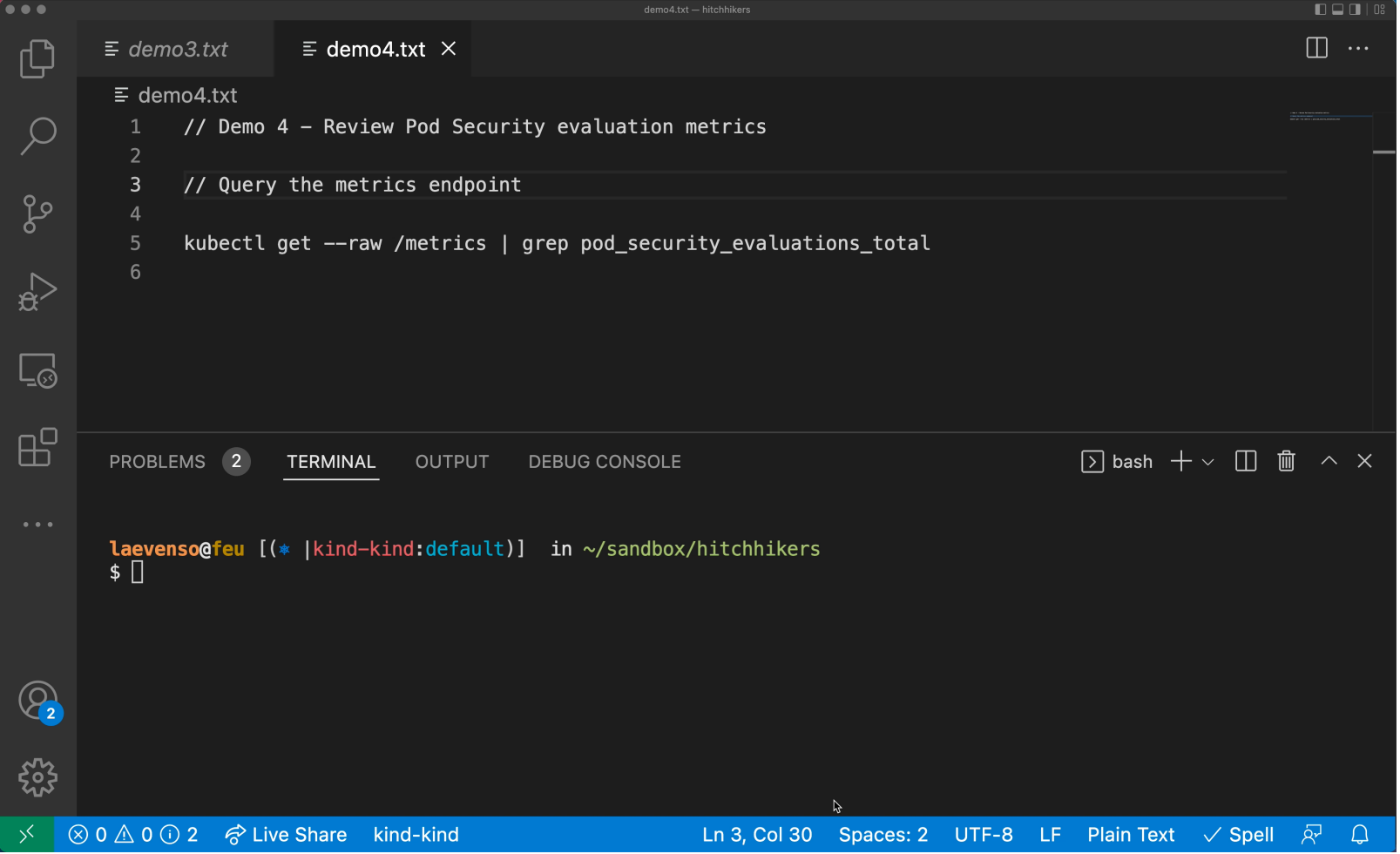
Privileged level and workload

@LachlanEvenson



Restricted level and workload

@LachlanEvenson



Review Pod Security evaluation metrics

@LachlanEvenson

Next Steps 🐬


@LachlanEvenson

Next Steps

- Go experiment to start learning!
- Use warn and audit on existing namespaces
    - Use dry-run to evaluate all resources in a namespace when applying an enforce label to a namespace
- Set warn to the same level as enforce so that users get early feedback
- Make a goal to get all workload namespaces to baseline standard
    - Start with audit and move towards enforce


Thanks for hitchhiking Pod Security with me! Now it's your turn…
So long, and thanks for all the fish! 🐬
Special thanks to Jim Angel, Tim Allclair, and Bridget Kromhout for their review!
https://bit.ly/psa-refs
@LachlanEvenson