

white duck

Hijack a Kubernetes Cluster – a Walkthrough

Microsoft
Partner



Gold Cloud Platform

Gold DevOps

Silver Application Development

Silver Security

Silver Application Integration

GitHub

Nico Meisenzahl



- Senior Cloud & DevOps Consultant at white duck
- Microsoft MVP, GitLab Hero
- Cloud Native, Kubernetes & Azure

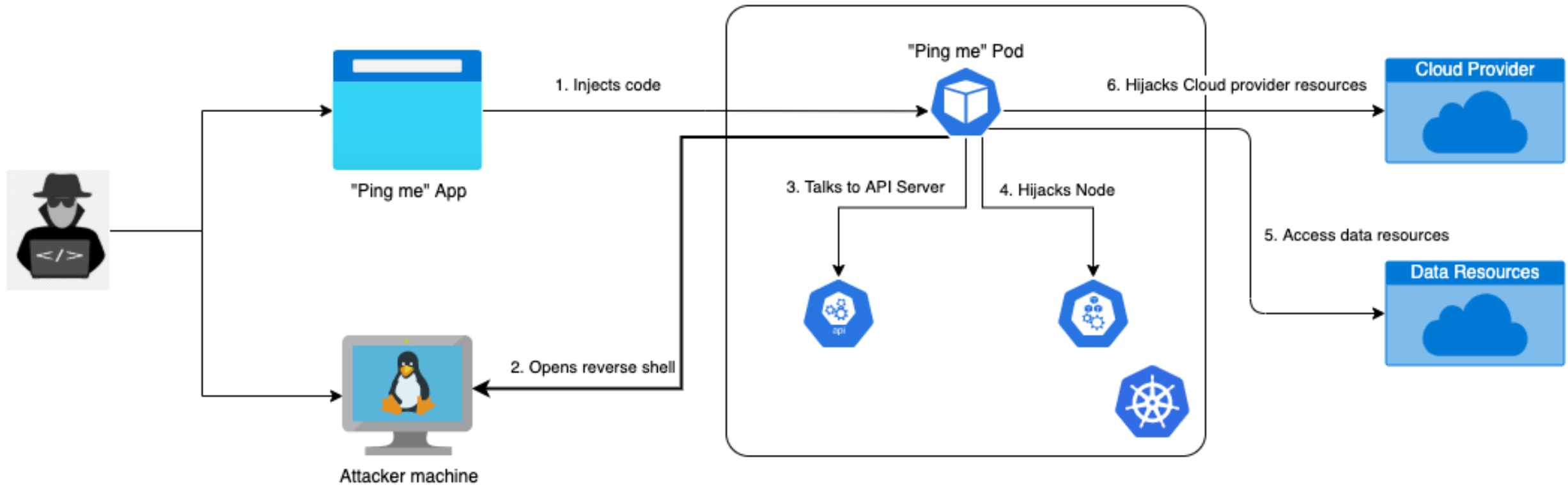
Phone: +49 8031 230159 0
Email: nico.meisenzahl@whiteduck.de
Twitter: [@nmeisenzahl](https://twitter.com/nmeisenzahl)
LinkedIn: <https://www.linkedin.com/in/nicomeisenzahl>
Blog: <https://meisenzahl.org>



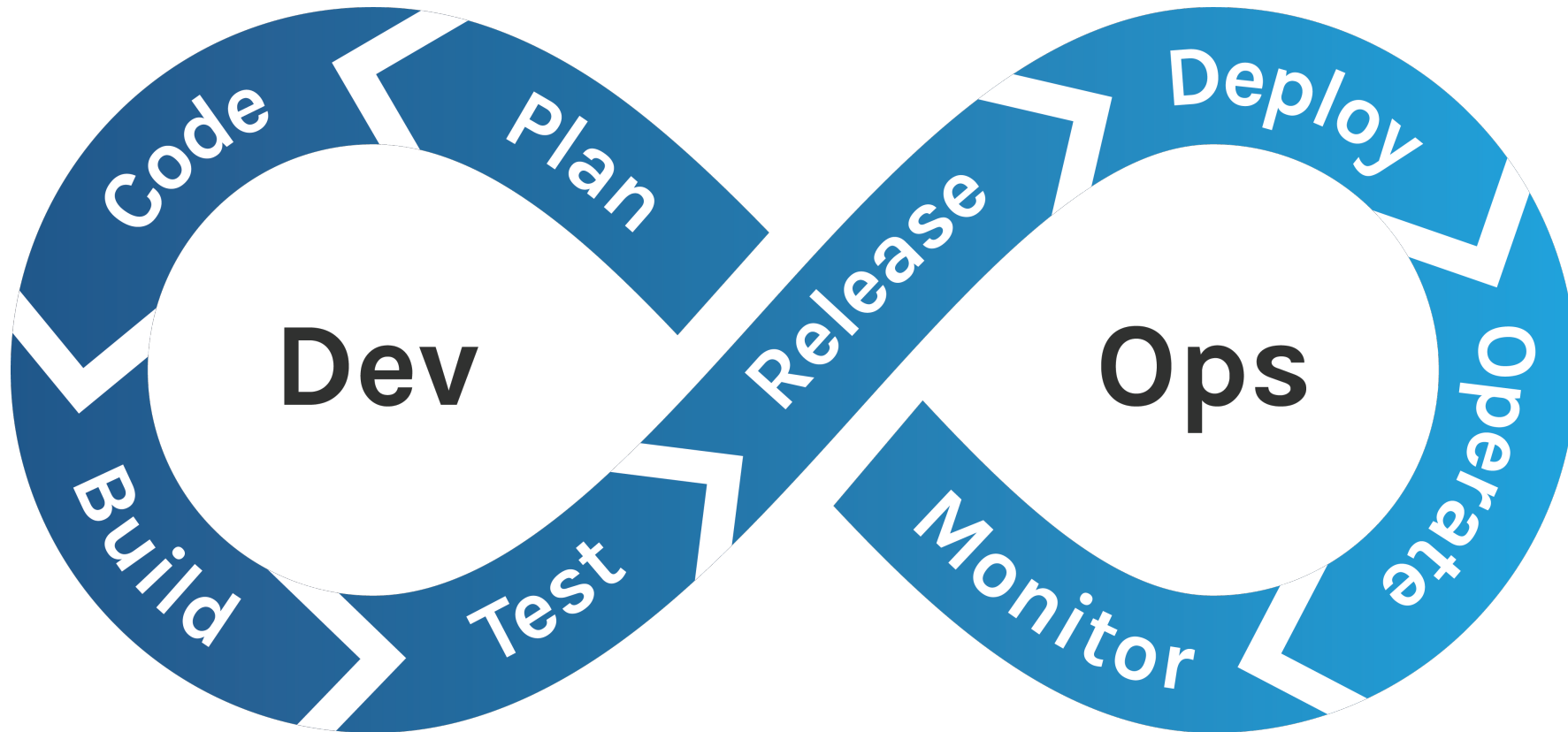
About this talk

- this is not an in-depth security talk
- it should make you aware of common attack vectors and how to prevent them
 - you will see demos on how to hijack a cluster
 - you will learn how to prevent those with common best practices
- one more slide, then we will start hijacking
 - <https://github.com/nmeisenzahl/hijack-kubernetes>

What we will do



Security quick wins through the DevOps cycle

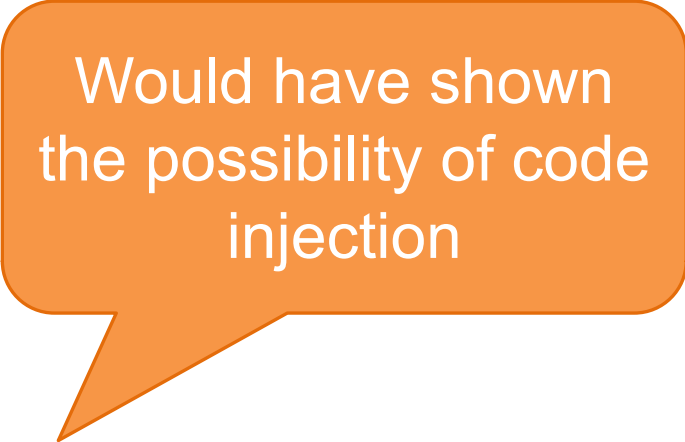


Ensure secure application code

- automate and enforce code checks
- schedule dependency scanning
 - e.g. Dependabot
- enforce Static Application Security Testing (SAST) in PRs
 - scans your code to identify potential security vulnerabilities
 - more details: https://owasp.org/www-community/Source_Code_Analysis_Tools

Ensure secure application code

- automate and enforce code checks
- schedule dependency scanning
 - e.g. Dependabot
- enforce Static Application Security Testing (SAST) in PRs
 - scans your code to identify potential security vulnerabilities
 - more details: https://owasp.org/www-community/Source_Code_Analysis_Tools



Would have shown
the possibility of code
injection

Build secure container images

- build secure/small container images – less is more
 - do only include required dependencies (no debugging tools!)
 - use self-contained binaries or “distroless” if possible
 - <https://github.com/GoogleContainerTools/distroless>
 - otherwise, use a small and secure Linux distro
- use and enforce SAST for validating your Dockerfiles
- scan your container images (on build and regularly)

Build secure container images

- build secure/small container images – less
 - do only include required dependencies (no dev)
 - use self-contained binaries or “distroless” if possible
 - <https://github.com/GoogleContainerTools/distroless>
 - otherwise, use a small and secure Linux distro
- use and enforce SAST for validating your Dockerfiles
- scan your container images (on build and regularly)

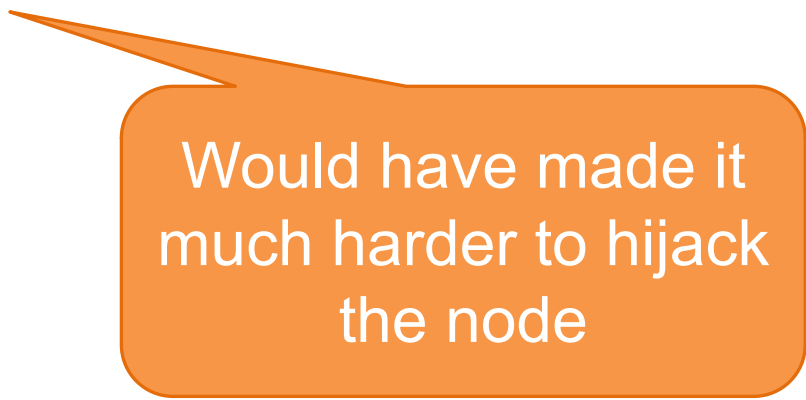
Would have made it much harder to hijack the container and further expend

Ensure secure deployment code

- as important as secure application code and Dockerfiles
- validate your deployment manifests using SAST
 - and enforce them via PRs
- can help you to implement best practices like denying
 - containers running as root
 - mounting hostPath
 - ...

Ensure secure deployment code

- as important as secure application code and Dockerfiles
- validate your deployment manifests using SAST
 - and enforce them via PRs
- can help you to implement best practices like denying
 - containers running as root
 - mounting hostPath
 - ...



Would have made it
much harder to hijack
the node

SAST Tooling

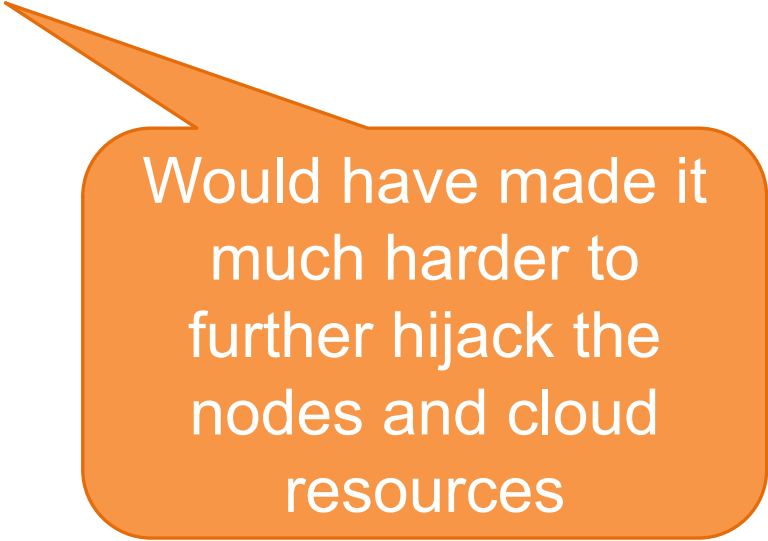
- Source code
 - <https://codeql.github.com>
 - <https://security-code-scan.github.io>
 - <https://securego.io>
- Kubernetes manifests
 - <https://kubesec.io>
 - <https://github.com/aquasecurity/trivy>
- Dockerfiles
 - <https://github.com/aquasecurity/trivy>
- Terraform
 - <https://github.com/tfsec/tfsec>
 - <https://github.com/aquasecurity/trivy>

Kubernetes policies

- enforce compliance and governance within clusters
 - verifying manifests is not enough!
- examples include enforcement of
 - read-only filesystems
 - denying hostPath mounts
 - denying containers running as root
 - ...

Kubernetes policies

- enforce compliance and governance within clusters
 - verifying manifests is not enough!
- examples include enforcement of
 - read-only filesystems
 - denying hostPath mounts
 - denying containers running as root
 - ...



Would have made it
much harder to
further hijack the
nodes and cloud
resources

Kubernetes policy Tooling

- Open Policy Agent Gatekeeper
 - <https://github.com/open-policy-agent/gatekeeper>
- Kyverno
 - <https://kyverno.io>
- Azure Policies
 - based on Open Policy Agent Gatekeeper
 - <https://docs.microsoft.com/azure/aks/use-azure-policy>

Network Policies

- granular deny or explicitly allow between containers and ingress/egress of the cluster
 - limit egress access to the internet
 - limit access between applications/namespaces
 - deny access to the Cloud provider metadata service
- <https://kubernetes.io/docs/concepts/services-networking/network-policies>

Network Policies

- granular deny or explicitly allow by ingress/egress of the cluster
 - limit egress access to the internet
 - limit access between applications/namespaces
 - deny access to the Cloud provider metadata service
- <https://kubernetes.io/docs/concepts/services-networking/network-policies>

Would have denied network connections (reverse shell, Redis, Internet, metadata service)

and

Container Runtime Security

- helps to detect malicious threads and workloads
 - untrusted process within container
 - a shell is running inside a container
 - container process mounting a sensitive path
 - a process making outbound network connections
- container runtime security tools like Falco can help
 - <https://github.com/falcosecurity/falco>

Container Runtime Security

- helps to detect malicious threads and v
 - untrusted process within container
 - a shell is running inside a container
 - container process mounting a sensitive path
 - a process making outbound network connections
- container runtime security tools like Falco can help
 - <https://github.com/falcosecurity/falco>

Would have detect all
our “work” within the
containers

Further best practises

- do not
 - share service accounts between applications
 - enable higher access levels for the default service account if not required
 - mount service account token if not required
 - <https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#use-the-default-service-account-to-access-the-api-server>
- review all third-party snippets before applying them
- implement a Web Application Firewall (WAF) to further secure your application

Further best practises

Wouldn't have
allowed us to talk to
the API server

- do not
 - share service accounts between applications
 - enable higher access levels for the default service account if not required
 - mount service account token if not required
 - <https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#use-the-default-service-account-to-access-the-api-server>
- review all third-party snippets before applying them
- implement a Web Application Firewall (WAF) to further secure your application

Would have denied
our code injection

Questions?



- Slides: <https://www.slideshare.net/nmeisenzahl>
- Demo: <https://github.com/nmeisenzahl/hijack-kubernetes>

Phone: +49 8031 230159 0
Email: nico.meisenzahl@whiteduck.de
Twitter: [@nmeisenzahl](https://twitter.com/nmeisenzahl)
LinkedIn: <https://www.linkedin.com/in/nicomeisenzahl>
Blog: <https://meisenzahl.org>

