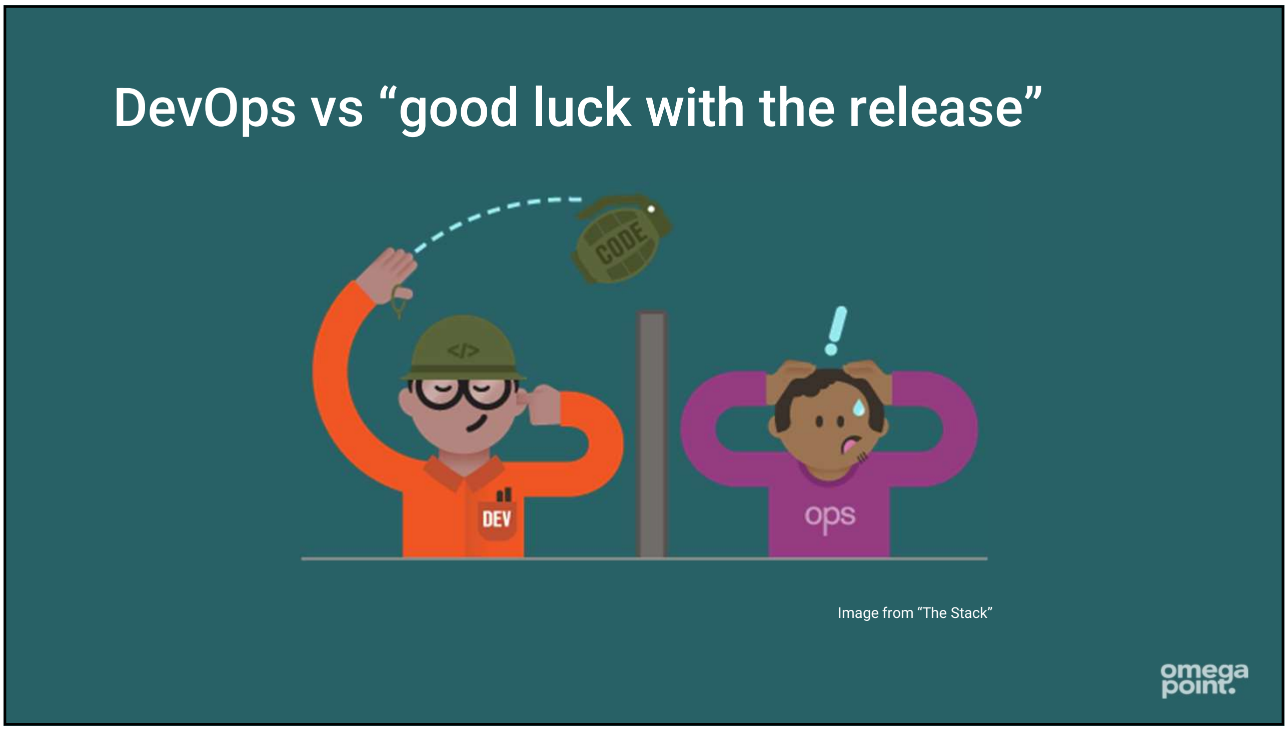
DevOps and Security @ OWASP

Who am I?

Mats Persson @ Omegapoint

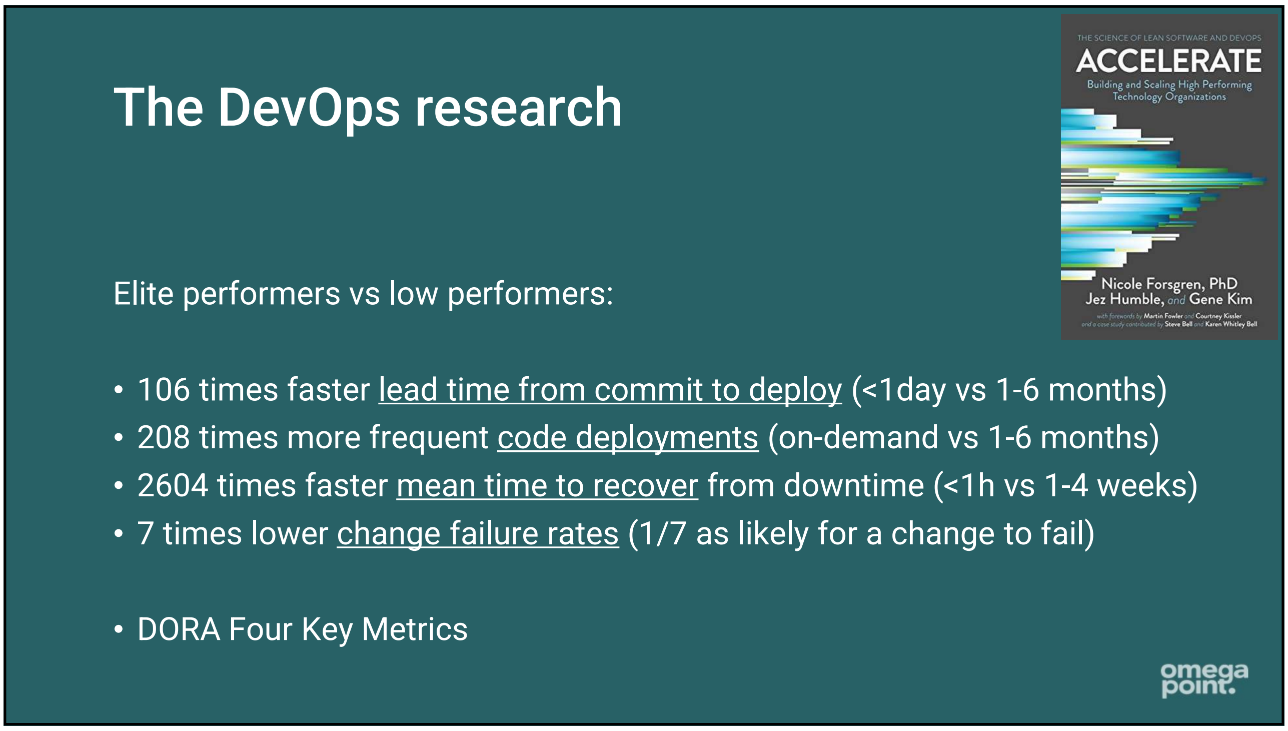
Secure Development
Modern Ways of Working
Security in the Cloud



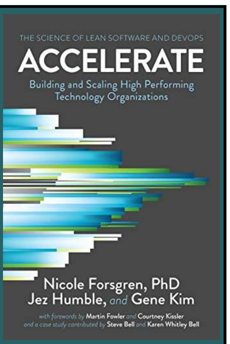
DevOps vs “good luck with the release”

Image from “The Stack”

omega point.



The DevOps research


THE SCIENCE OF LEAN SOFTWARE AND DEVOPS
ACCELERATE
Building and Scaling High Performing Technology Organizations
Nicole Forsgren, PhD
Jez Humble, and Gene Kim

Elite performers vs low performers:

- 106 times faster lead time from commit to deploy (<1day vs 1-6 months)
- 208 times more frequent code deployments (on-demand vs 1-6 months)
- 2604 times faster mean time to recover from downtime (<1h vs 1-4 weeks)
- 7 times lower change failure rates (1/7 as likely for a change to fail)

- DORA Four Key Metrics

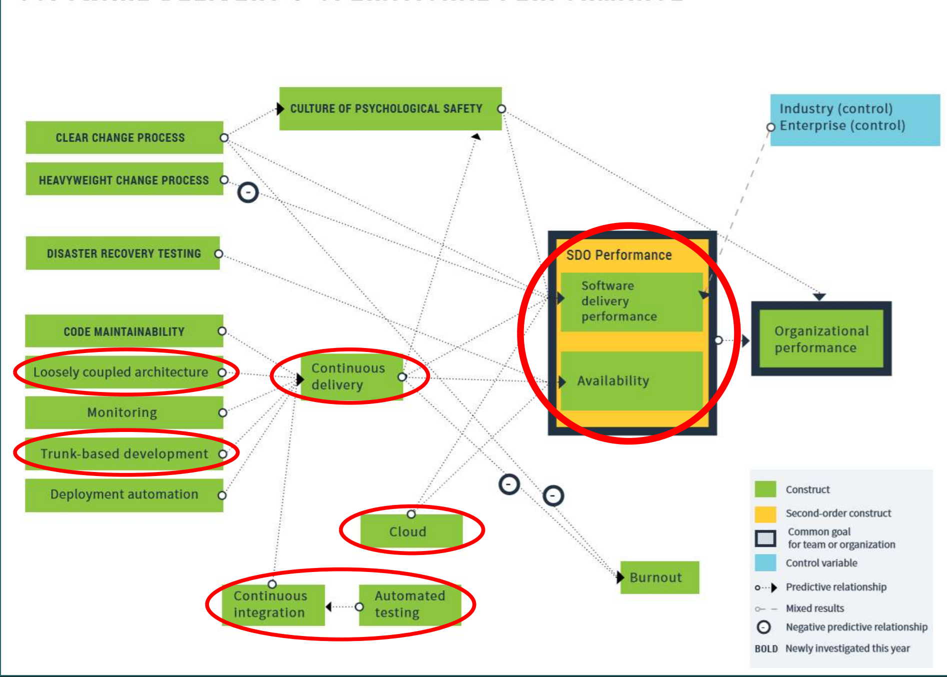
omega point.

The DevOps paradox

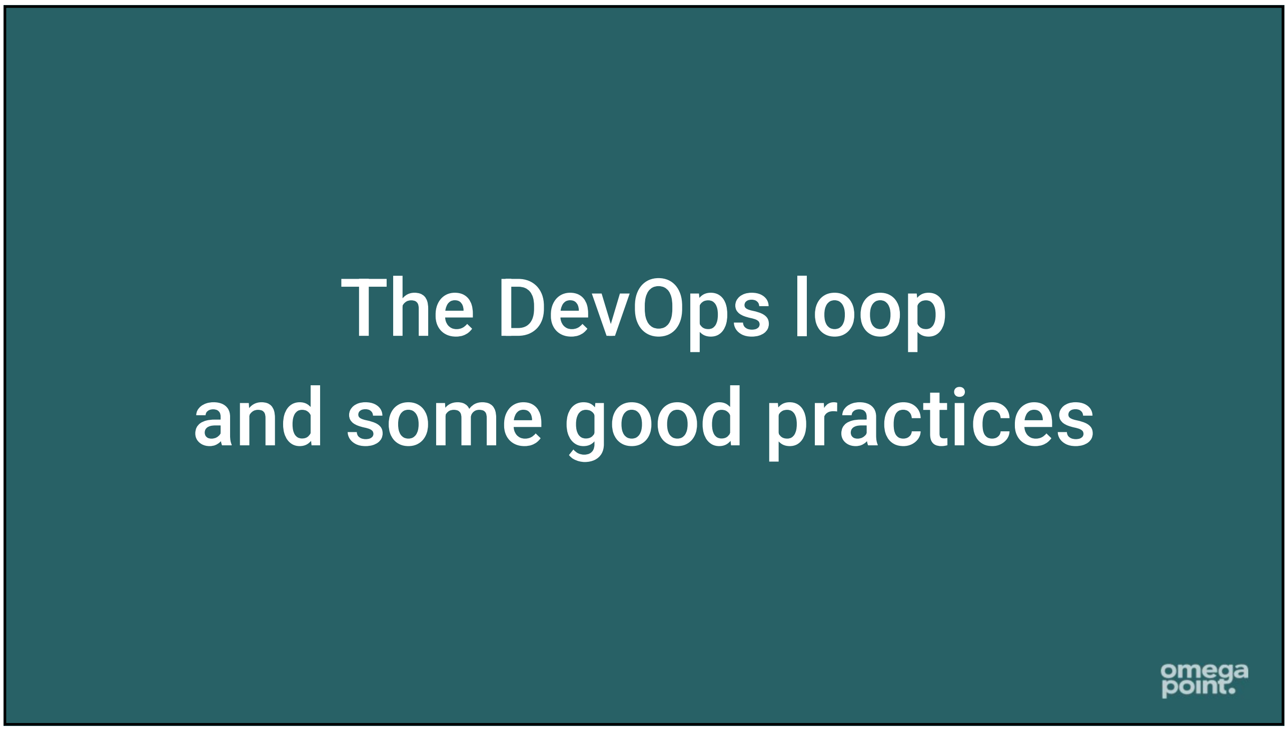
High performers deliver more, faster, and with higher stability

omega point.

SOFTWARE DELIVERY & OPERATIONAL PERFORMANCE

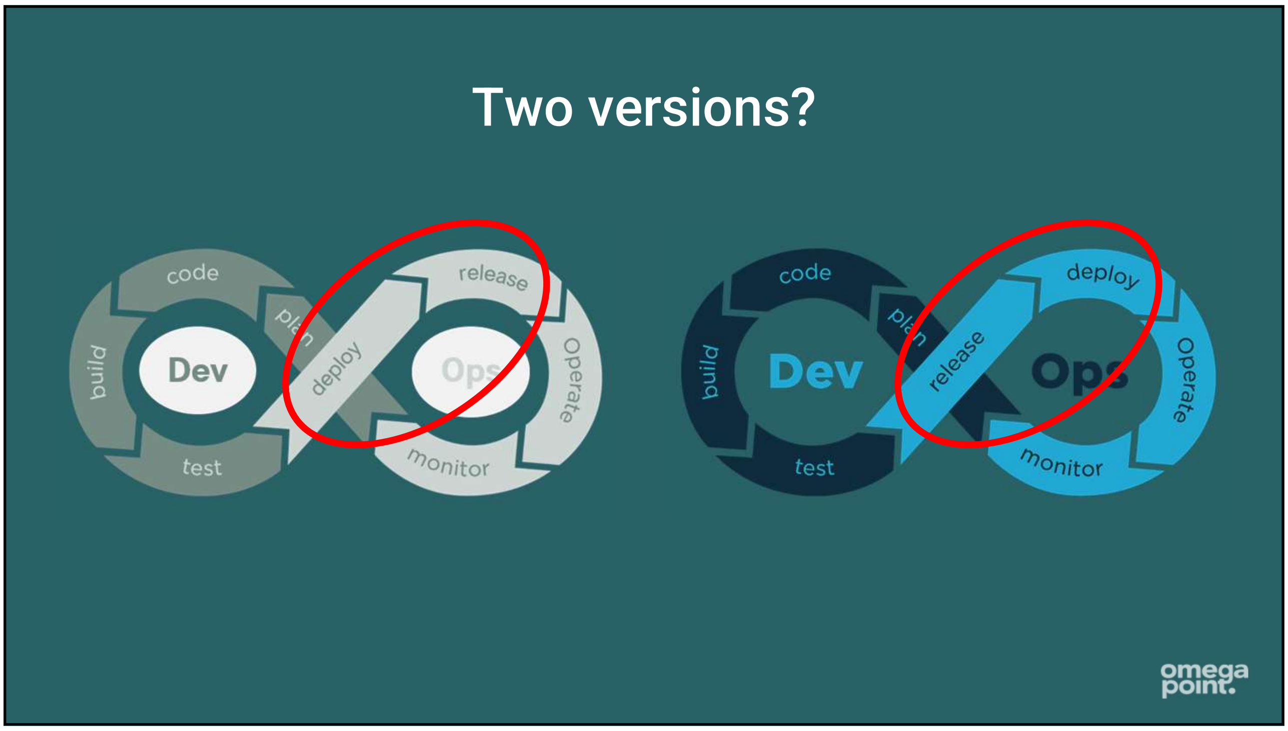


omega point.

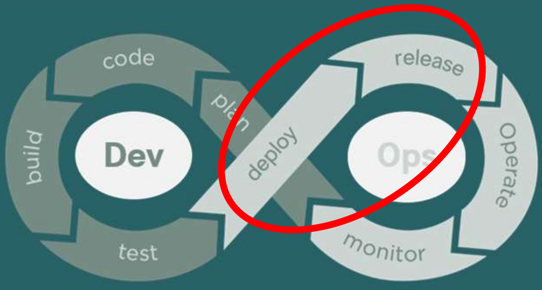


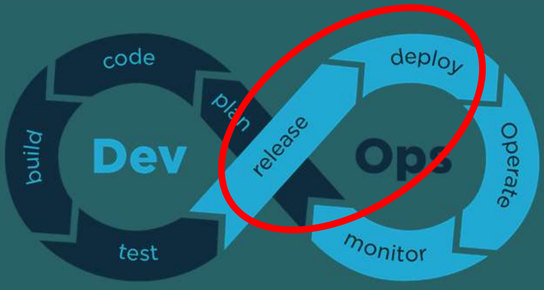
The DevOps loop and some good practices

omega point.

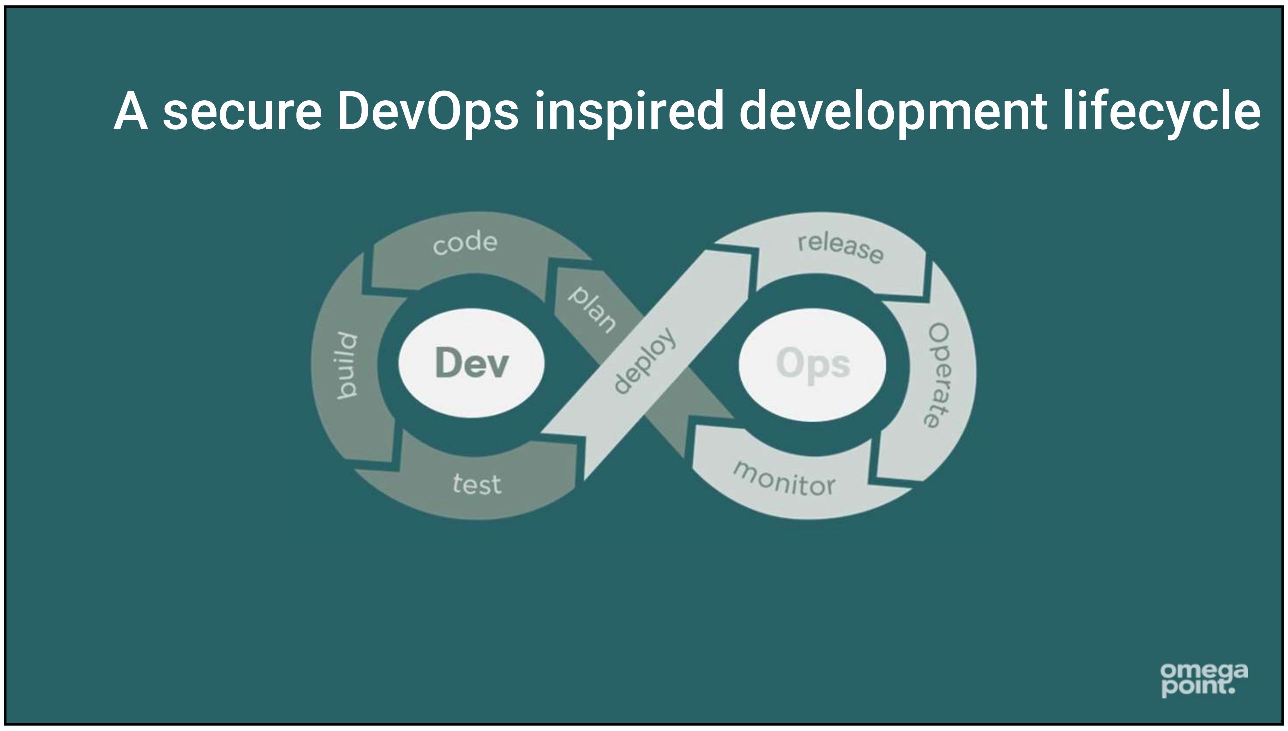


Two versions?

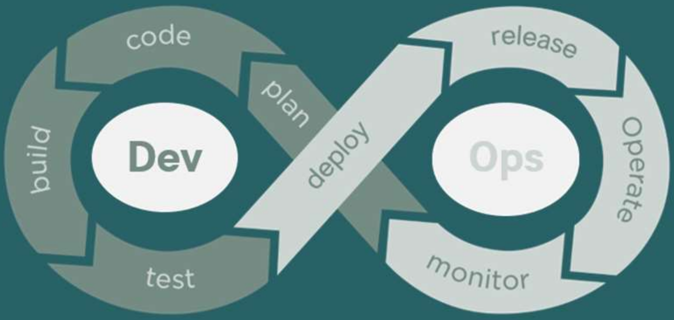




omega point.

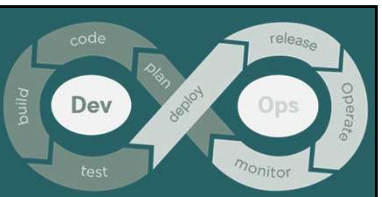


A secure DevOps inspired development lifecycle

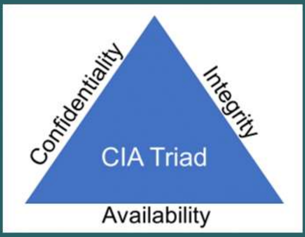


omega point.

Plan



- Consider confidentiality, integrity, and availability (C-I-A) requirements for your system or application:
  - Confidentiality - prevent unauthorized disclosure of information
  - Integrity – prevent unauthorized modification of information
  - Availability – ensure information is available when needed
- Discuss in the team what is important for you
- Don’t forget the Privacy aspects



omega point.

Plan

- Work with all 4 work item types
  - Features
  - Defects
  - Risk (regulatory, security, compliance)
  - Technical Debt (old software, architecture, test/build automation)

- Recommended to spend 20% time to limit tech debt (I would include Risk)

Plan

Features

Risks/Debts

Defects

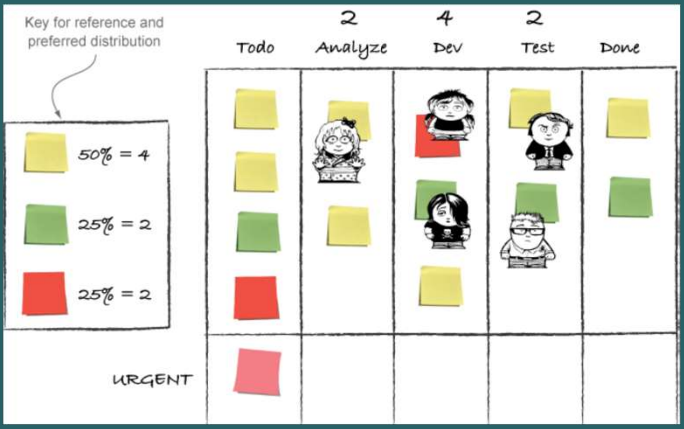
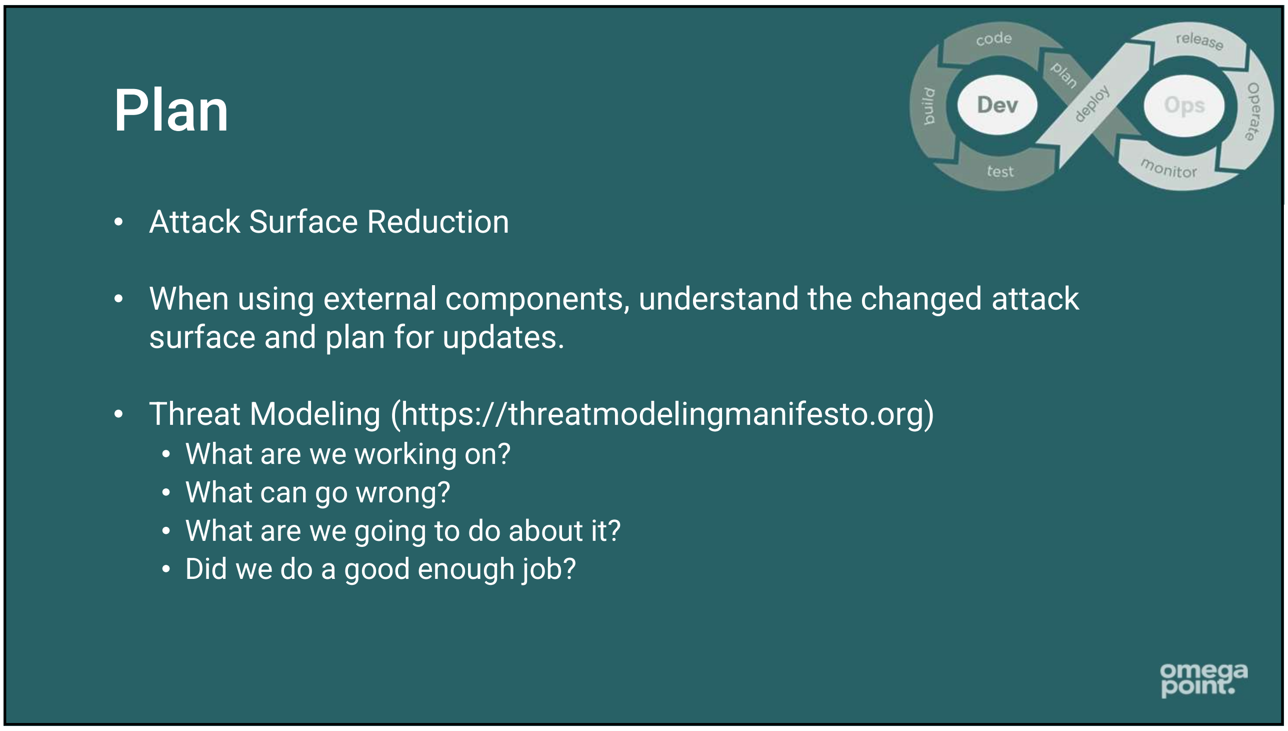
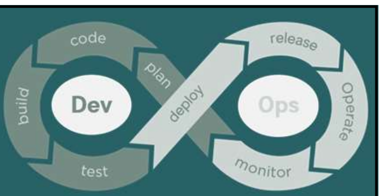


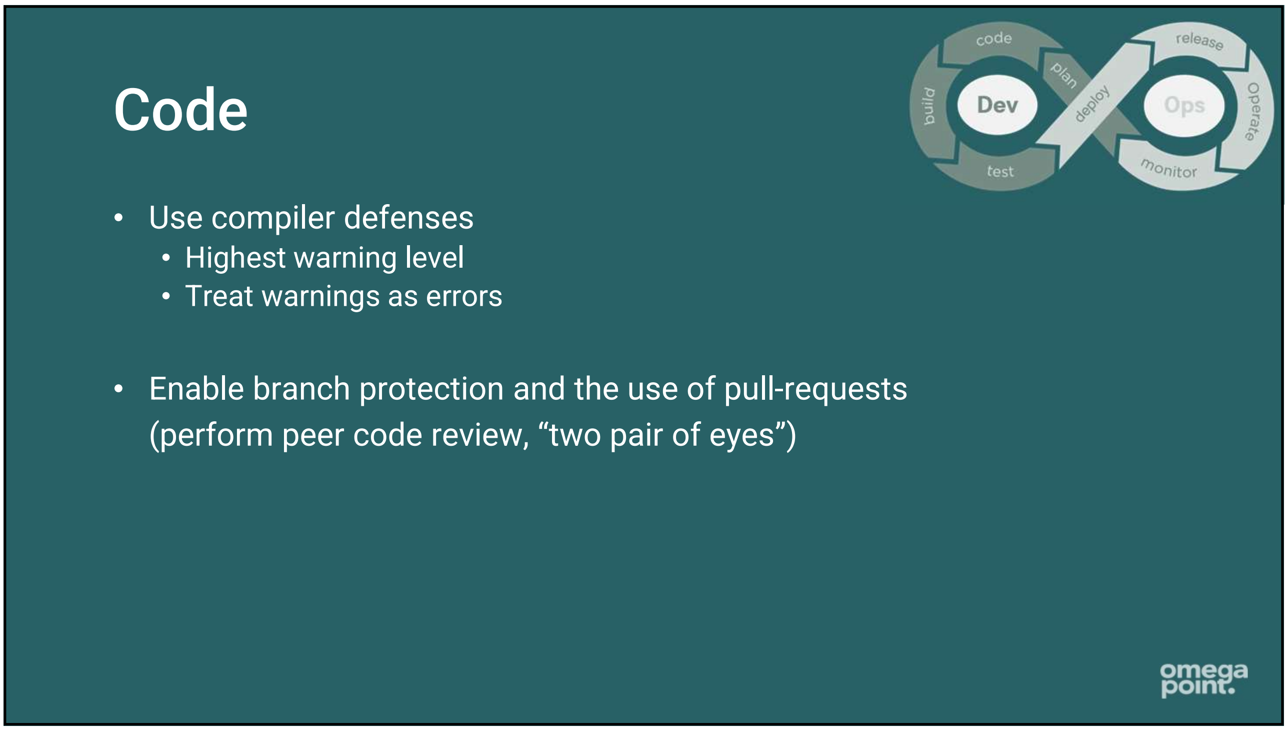
Image from the book "Kanban in Action"



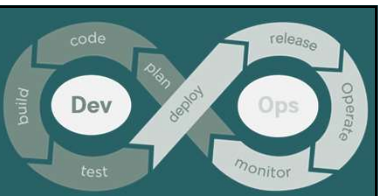
Plan



- Attack Surface Reduction
- When using external components, understand the changed attack surface and plan for updates.
- Threat Modeling (https://threatmodelingmanifesto.org)
  - What are we working on?
  - What can go wrong?
  - What are we going to do about it?
  - Did we do a good enough job?

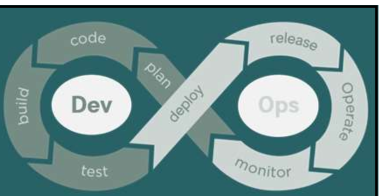


Code



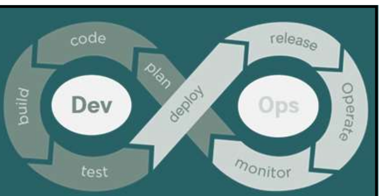
- Use compiler defenses
  - Highest warning level
  - Treat warnings as errors
- Enable branch protection and the use of pull-requests (perform peer code review, “two pair of eyes”)

Code



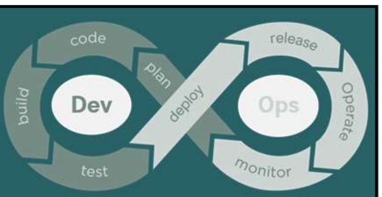
- Secrets Management, API-keys, credentials for different environments
- Remember logging and traceability (for your own sake)
- Do input validation and output encoding
- Write unit tests (consider Test Driven Development)

Build



- Continuous Integration (daily), trunk-based development
- Verify use of external components (create Software Bill of Material, SBoM)
  Might require tooling support (e.g. OWASP Dependency Check, Snyk)
- Scan the code you write with a static code analysis tool (SAST)
  (e.g. Synopsys Coverity, GitHub Advanced Security)
- Scanning external components (SCA) may be done in every build.
  Static scan of the code (SAST) might require nightly builds.

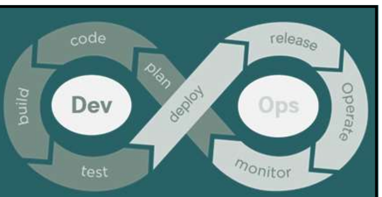
Test



- Automate test cases (to build confidence in your releases)
- Fuzz testing (send invalid, unexpected, or random data as input)
- Derive test cases from Threat Modeling

omega point.

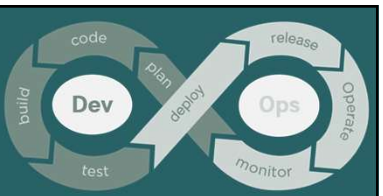
Deploy



- Continuous Delivery – make every build releasable and maybe even deployed to the production environment.
- Consider using feature toggles to enable/disable new features or not yet complete features from being released to customers.
- Make sure all environments are production like (infrastructure as code, easier done in the cloud)
- Consider using A/B testing to verify if new features deliver the intended value to customers.

omega point.

Release



- Release on demand. Depending on business requirements (or maturity) releasing to customers might be a manual or automatic step.

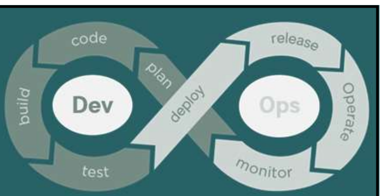
- Releasing often helps to reduce risk.

  You practice the release process which will make releasing less painful and the difference between releases lessen:

  “If it hurts, do it more often”

omega point.

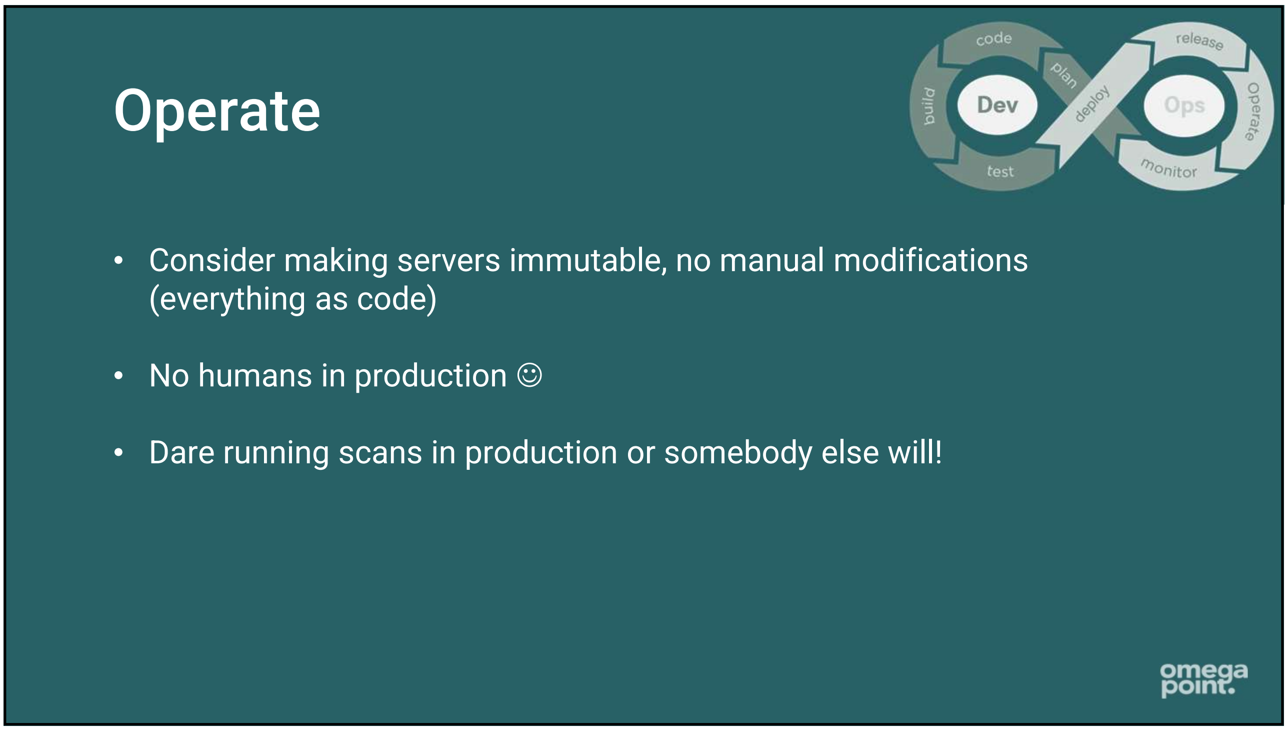
Release



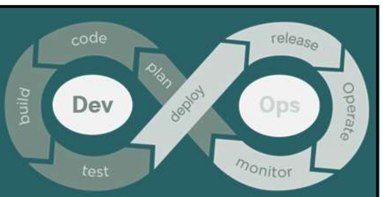
- Release to customers without downtime and during daytime when everybody involved is available at work.

- Use Blue/Green or Canary releases and/or feature toggles to release new features to customers.

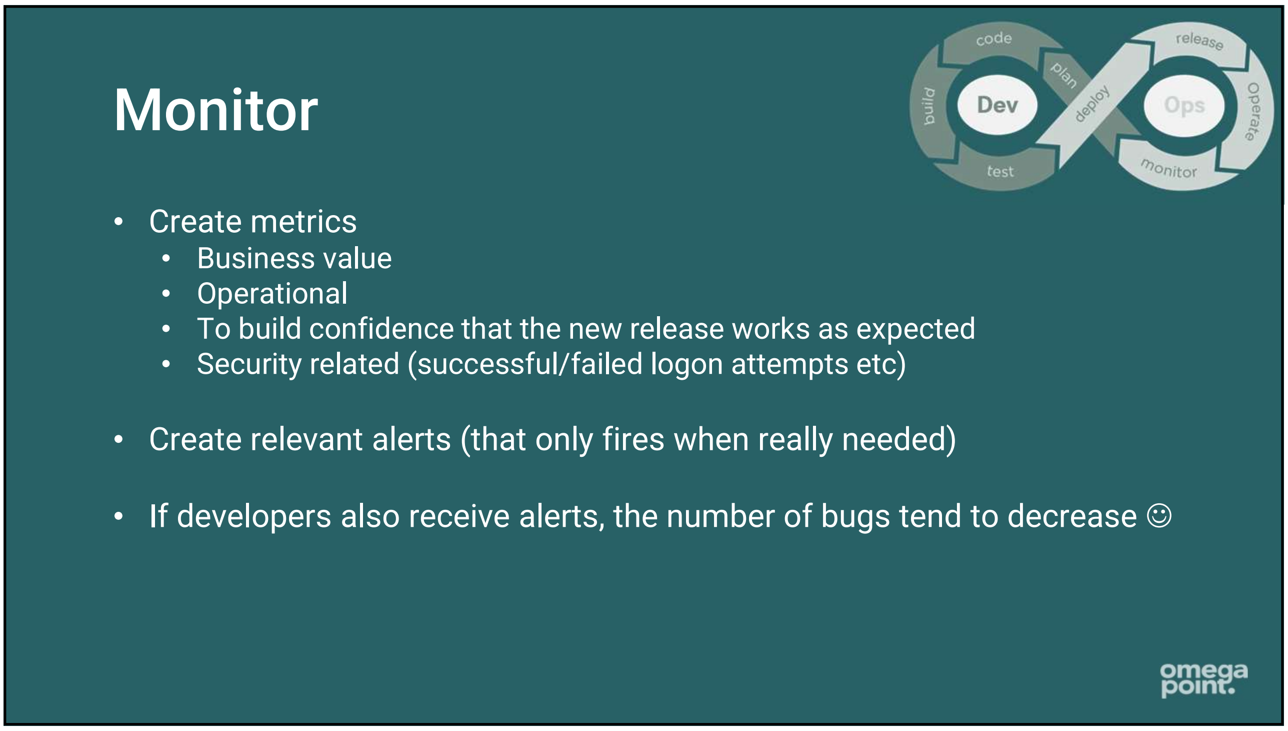
omega point.



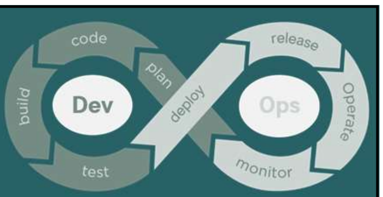
Operate



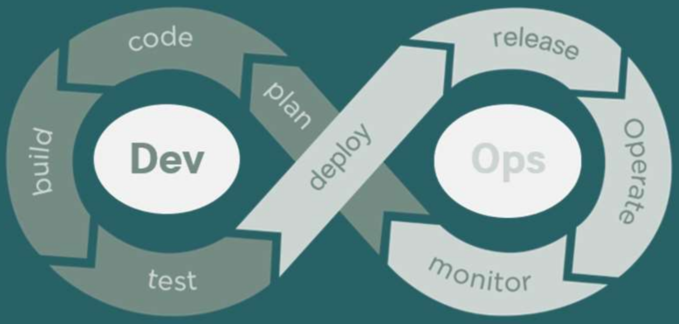
- Consider making servers immutable, no manual modifications (everything as code)
- No humans in production ☺
- Dare running scans in production or somebody else will!



Monitor



- Create metrics
  - Business value
  - Operational
  - To build confidence that the new release works as expected
  - Security related (successful/failed logon attempts etc)
- Create relevant alerts (that only fires when really needed)
- If developers also receive alerts, the number of bugs tend to decrease ☺



omega point.

omegapoint.se/devops

omega point.

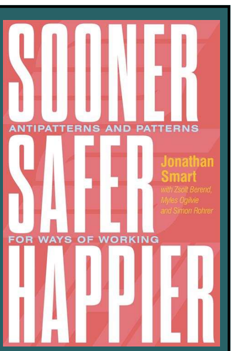
Book Tip

Sooner, Safer, Happier

Antipatterns and Patterns for Business Agility

Better - Value - Sooner - Safer - Happier

By Jonathan Smart
(2020)


SOONER
ANTIPATTERNS AND PATTERNS
SAFER
Jonathan Smart
with Zsolt Berend, Myles Ogilvie and Simon Rohrer
FOR WAYS OF WORKING
HAPPIER

omega point.