How to Design a Secure Serverless Architecture

2023 Update

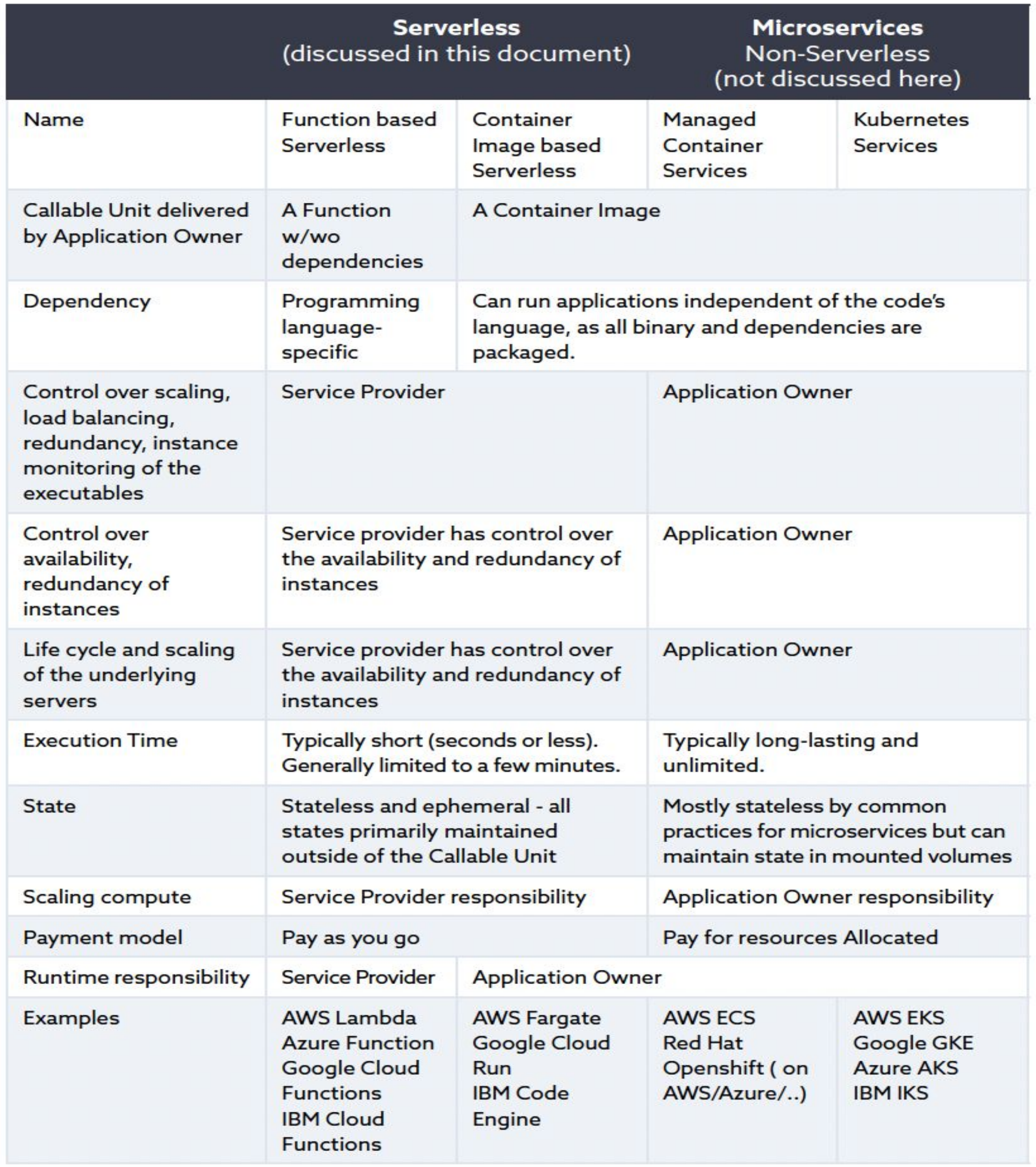
Release Date: 10/24/2023

Serverless Working Wroup

How to Design a Secure Serverless Architecture

Updated 2023 Version

Document Table of Contents



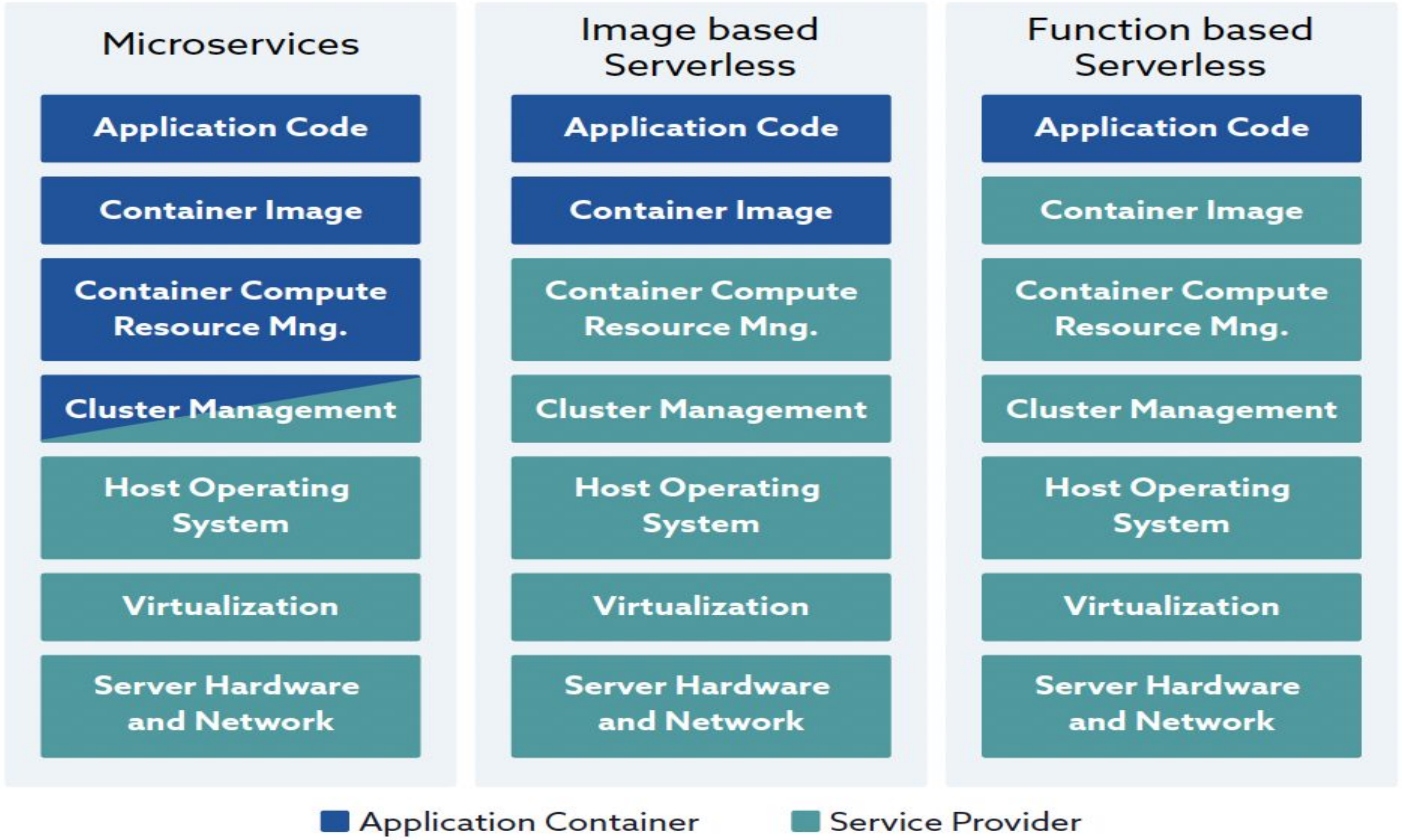
	Serverless (discussed in this document)		Microservices Non-Serverless (not discussed here)	
Name	Function based Serverless	Container Image based Serverless	Managed Container Services	Kubernetes Services
Callable Unit delivered by Application Owner	A Function w/wo dependencies	A Container Image		
Dependency	Programming language-specific	Can run applications independent of the code's language, as all binary and dependencies are packaged.		
Control over scaling, load balancing, redundancy, instance monitoring of the executables	Service Provider		Application Owner	
Control over availability, redundancy of instances	Service provider has control over the availability and redundancy of instances		Application Owner	
Life cycle and scaling of the underlying servers	Service provider has control over the availability and redundancy of instances		Application Owner	
Execution Time	Typically short (seconds or less). Generally limited to a few minutes.		Typically long-lasting and unlimited.	
State	Stateless and ephemeral - all states primarily maintained outside of the Callable Unit		Mostly stateless by common practices for microservices but can maintain state in mounted volumes	
Scaling compute	Service Provider responsibility		Application Owner responsibility	
Payment model	Pay as you go		Pay for resources Allocated	
Runtime responsibility	Service Provider	Application Owner		
Examples	AWS Lambda Azure Function Google Cloud Functions IBM Cloud Functions	AWS Fargate Google Cloud Run IBM Code Engine	AWS ECS Red Hat Openshift (on AWS/Azure/..)	AWS EKS Google GKE Azure AKS IBM IKS

Serverless:

A cloud-computing execution model in which the cloud provider is responsible for allocating compute and infrastructure resources needed to serve Application Owners› workloads.

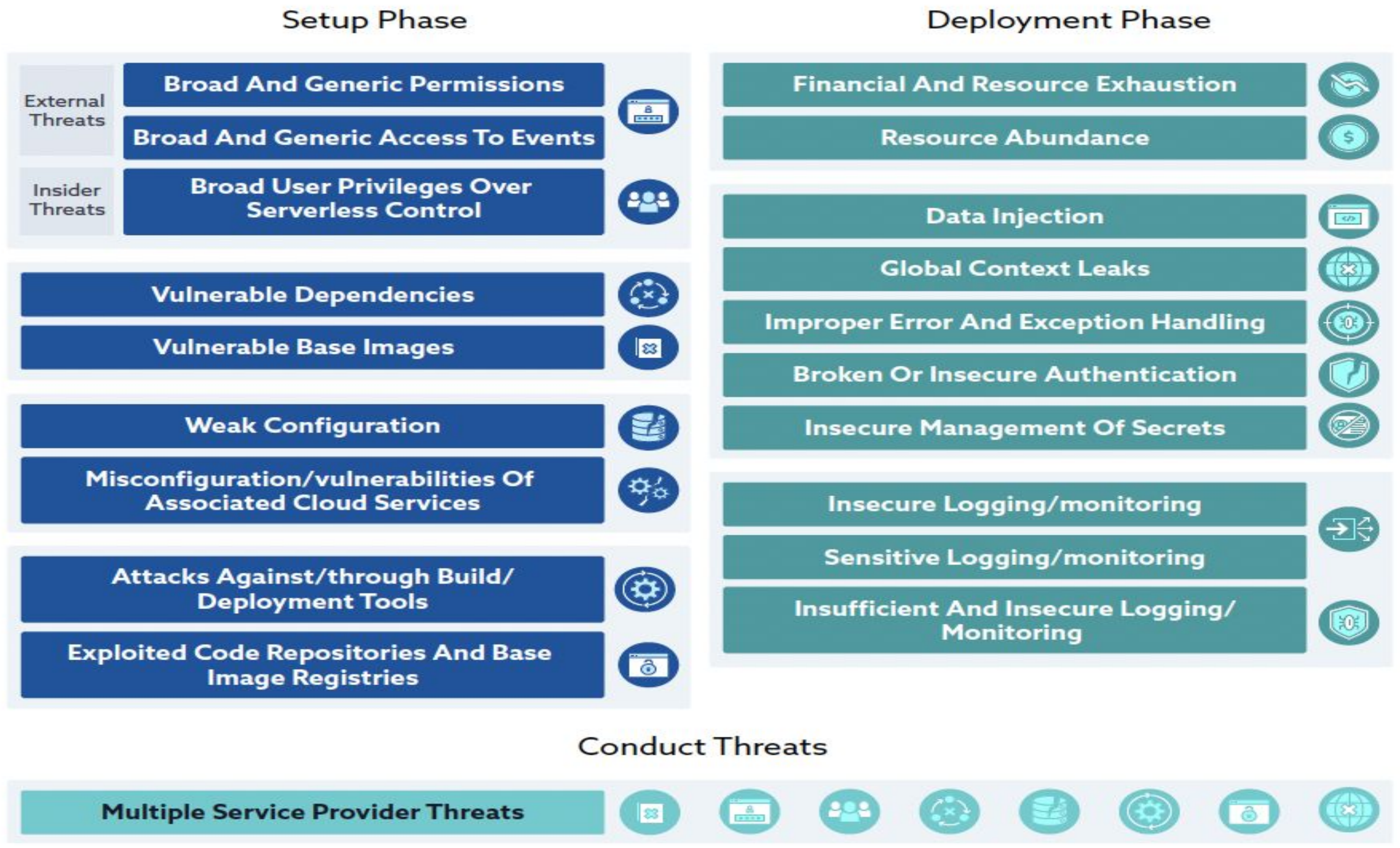
An Application Owner is no longer required to determine and control how many compute resources (and at what size) are allocated to serve their workload at any given time. It can rely on an abundance of compute resources that will be available to serve the workload on-demand.

Therefore, serverless computing is offered under a "Pay as you go" paradigm where payment is generally made on actual physical resources like central processing unit (CPU) usage time.



Comparative Shared responsibility model

The Threat Landscape



Setup Phase

- External Threats
  - Broad And Generic Permissions
  - Broad And Generic Access To Events
- Insider Threats
  - Broad User Privileges Over Serverless Control
- Vulnerable Dependencies
- Vulnerable Base Images
- Weak Configuration
- Misconfiguration/vulnerabilities Of Associated Cloud Services
- Attacks Against/through Build/ Deployment Tools
- Exploited Code Repositories And Base Image Registries

Deployment Phase

- Financial And Resource Exhaustion
- Resource Abundance
- Data Injection
- Global Context Leaks
- Improper Error And Exception Handling
- Broken Or Insecure Authentication
- Insecure Management Of Secrets
- Insecure Logging/monitoring
- Sensitive Logging/monitoring
- Insufficient And Insecure Logging/ Monitoring

Conduct Threats

- Multiple Service Provider Threats

The Threat Landscape

25 Serverless Threats for Application Owners

Sr. No.	Threat Summary	Threat Description	Mitigations (Security Controls)
Application Owner Setup Phase Threats (A)			
Unique to Serverless			
1.	**Broad and generic permissions** Not maintaining the least privilege principle for Callable Units.	Application Owners may define the set of privileges that each Serverless Callable Unit will have while running. Excessive permissions can be taken advantage of as part of an attack.	See section 6, 6.4.5, 6.4.7.1, 6.4.7.2. etc.
2.	**Broad and generic access to Events** Not maintaining the least privilege principle for the initiation of events triggering the Callable Unit.	Application Owners may define who may initiate Events that will trigger the Callable Units. Broader access greatly simplifies the execution of attacks. Especially in event-driven serverless architectures, this has an impact on the attack surface.	See section 6, 6.4.5, 6.4.7.1, 6.4.7.2. etc.
3.	**Broad user privileges over serverless control** Not maintaining the least privilege principle for the DevOps team.	Application Owners may define who may have access to set up the Serverless service, image store, etc. Broader access adds additional potential paths for attackers and increases risks from insiders.	See section 6, 6.4.5, 6.4.7.1, 6.4.7.2. etc.
……	………	……….	………

Design Considerations for Serverless

Application Architects have to be cognizant of both inherent weaknesses present in Serverless technology vs. weaknesses they can introduce in their design. Understanding these weaknesses will give them a better grasp of the security controls required to be implemented.

Serverless architecture has many benefits, even from a security perspective, that are weighed as part of Secure microservices design considerations. Some of those benefits include:

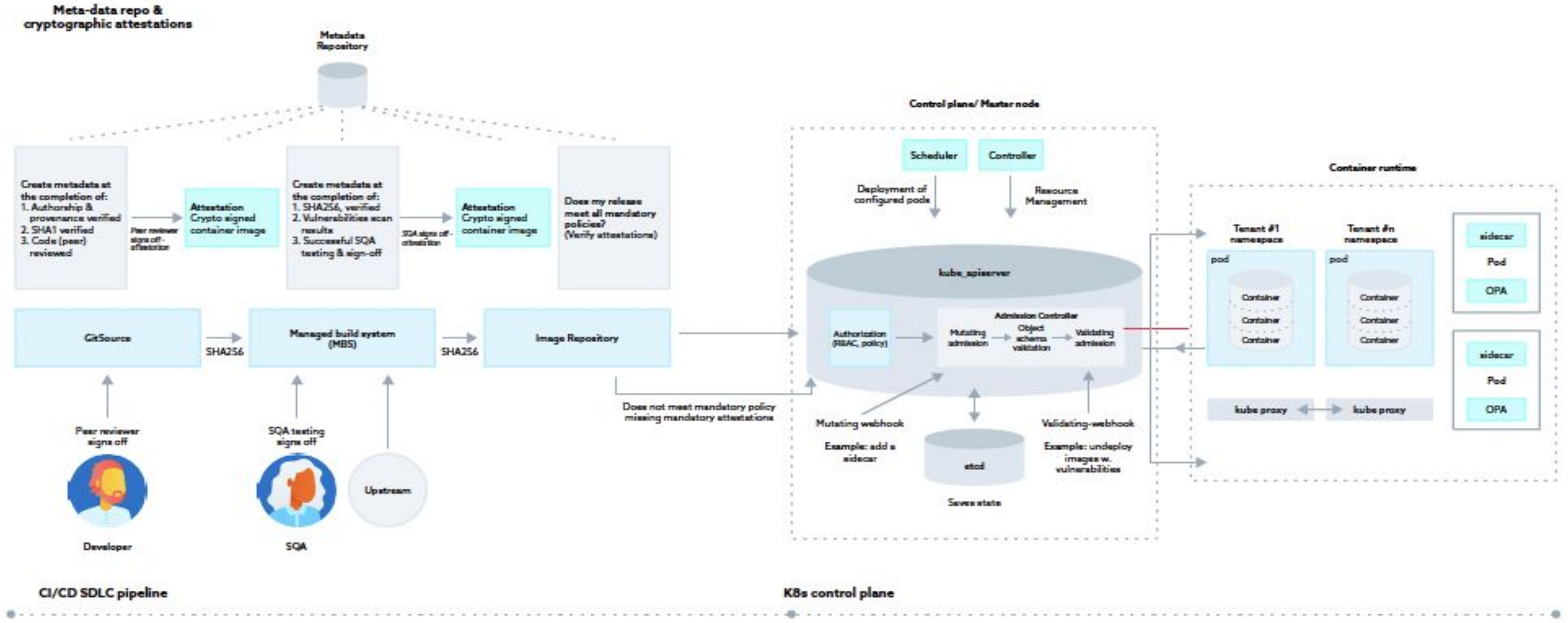
1. Stateless and Ephemeral: Short-lived serverless functions are processing unencrypted data in memory for a short period of time. Serverless functions do not write to a local disk. Hence, functions that need to persist state rely on external data stores, thus reducing the likelihood of exploits by attacks designed for long-lived targets.

2. Each serverless function requires only a subset of data to perform its micro-focused service. So as long as this function has the correct permission to access only the data it requires, then a successful exploit of a function should be more focused on what data it can potentially exfiltrate.

3. Serverless applications run within containers managed by a CSP or within self-managed containers. Hence, they have some inherent security benefits of containers that run on immutable container images. Containers that do not require long-lived servers can easily be assigned continuously to patch container images and compute instances. Lessening concerns of running on Vulnerable or Unpatched underlying infrastructure.

CI-CD Pipelines, Function Code, Code Scans and Policy Enforcement for Functions and Containers

	Planning	Planning	Planning	Planning	Planning	Planning	Planning	Planning
Security Controls	• System design • Business process • Policy, procedure • Minimum viable product (MVP) • Risk assessment • Requirements gathering	• Application devleopment • Source code repository to store code, configuration, policy, YAML • Unit test • Code Review • Static analysis/SAST Secure coding	• Container images • Private registry • Build managment • CI/CD management • Configuration	• Integration testing • Dynamic analysis security testing (DAST) • Acceptance test • Penetration test	• Generate checksum digital signature for the release package • Binaries, base container image, configuration, scripts • MVP baselines • CI/CD orchestrator • Artifact repository	• Virtualization engine • K8s platform lifecycle • Service mesh • Identity and access management • Platform data • Deployment policies • Configuration • ETCD	• Cluster management • Load balancing • System scaling • Rate limiting • Backup management • Operation dashboard • Container isolation • Application access and data • Observability • Container image scanning	• Logging events for user, network, application and data • Log analysis auditing • Operations monitoring • Alerting and notification • Log aggregation & archival
Vulnerabilities	• Insecure untended data caching • Secure design • Incorporate AuthN, AuthZ via API gateways	• Data injection • OWASP top 10 • Improper exception handling	• Insecure/weak configuration • API OWASP top 10	• Included 3rd Party • Library dependencies	• Insecure/weak configuration • API OWASP top 10 • Attacks against automated deployment work • Exploited image repositories	• Broad & generic permissions • Broken authentication • Insecure management of secrets, traffic (at rest, in motion)	• Vulnerable images • Broad & generic permissions • Broken authentication API OWASP top 10 • Portal vulnerabilities	• Insufficient logging and monitoring • Embedded malware



Meta-data repo & cryptographic attestations
Metadata Repository
Create metadata at the completion of: 1. Authorship & provenance verified 2. SHA1 verified 3. Code (peer) reviewed
Peer reviewer signs off - attestation
Attestation Crypto signed container image
Create metadata at the completion of: 1. SHA256, verified 2. Vulnerabilities scan results 3. Successful SQA testing & sign-off
SQA signs off - attestation
Attestation Crypto signed container image
Does my release meet all mandatory policies? (Verify attestations)
GitSource
SHA256
Managed build system (MBS)
SHA256
Image Repository
Does not meet mandatory policy missing mandatory attestations
Peer reviewer signs off
Developer
SQA testing signs off
SQA
Upstream
CI/CD SDLC pipeline
Control plane/ Master node
Scheduler
Controller
Deployment of configured pods
Resource Management
kube_apiserver
Authorization (RBAC, policy)
Admission Controller
Mutating admission
Object schema validation
Validating admission
Mutating webhook
Example: add a sidecar
etcd
Saves state
Validating-webhook
Example: undeploy images w. vulnerabilities
K8s control plane
Container runtime
Tenant #1 namespace
Tenant #n namespace
pod
pod
Container
Container
Container
Container
Container
Container
kube proxy
kube proxy
sidecar
Pod
OPA
sidecar
Pod
OPA

Serverless Best Practices

- Regular Risk Evaluation Cycles
- Vendor “Well Architected” Reference Documentation
- Inspection of Upstream Identity Providers
- Minimization of Durable Assumptions and Authorizations
- Usage of Managed Identities and/or Keys
- Usage of Appropriate Data Protection Practices
- Logs Protection for Serverless Applications
- Secure Communication Channels
- Continuous Security Testing and Remediation
- Serverless Incident Response Plan
- Secure Serverless Supply Chain
- Serverless Application Performance Monitoring
- Continuous Security Testing and Remediation

Serverless Advances for Data Privacy

1. **Self-protection:** functions will, in real-time, evaluate and adapt the security and micro-segmentation around each resource, even by using predictive analytics(AI/ML) to evaluate the security posture and define new alerting mechanisms.

2. **Pre-defined compliant functions**: A set of functions that developers could leverage and use, according to the guardrails established and type of data required to be used. This could be a natural extension to set up guardrails, a kind of catalog for serverless. This would have the benefits of creating guardrails without reducing agility.

3. **Secure enclaves for FaaS**: With the scrutiny and assurance that some regulated industries want from the Cloud environments, organizations need to leverage mechanisms to be compliant with ever increasing regulations.

4. **Zero-Knowledge Proofs for Secure Function Execution**: In the future, serverless platforms may incorporate advanced cryptographic techniques like zero-knowledge proofs to ensure the privacy and security of data during function execution. With zero-knowledge proofs, the serverless platform can verify the correctness of a computation without needing to know the actual data being processed.

Serverless working group

CSA page:
https://cloudsecurityalliance.org/research/working-groups/serverless/

Circle community:
https://circle.cloudsecurityalliance.org/community-home1?communitykey=5a79a42c-d173-49da-a4a3-47c27b7f552c

Other publications:

- **FaaS Serverless Control Framework (Set) based on NIST 800-53 R5 controls**
- **C-Level Guidance to Securing Serverless Architectures**