2021

# Cloud Computing Security

DANNIES PAHLEVI

# Contents

## Module objectives

- Understanding Cloud Computing Concept
- Overview of Container Technology and Serverless Computing
- Understanding Cloud Computing Threats
- Understanding Cloud Hacking
- Understanding Cloud Computing Security
- Overview of Various Cloud Computing Security Tools

# Module Flow – Cloud Computing Concepts


## Introduction to cloud computing

Cloud computing is an on-demand of IT capabilities where IT infrastructure and applications are provided to subscribes as a metered service over a network.

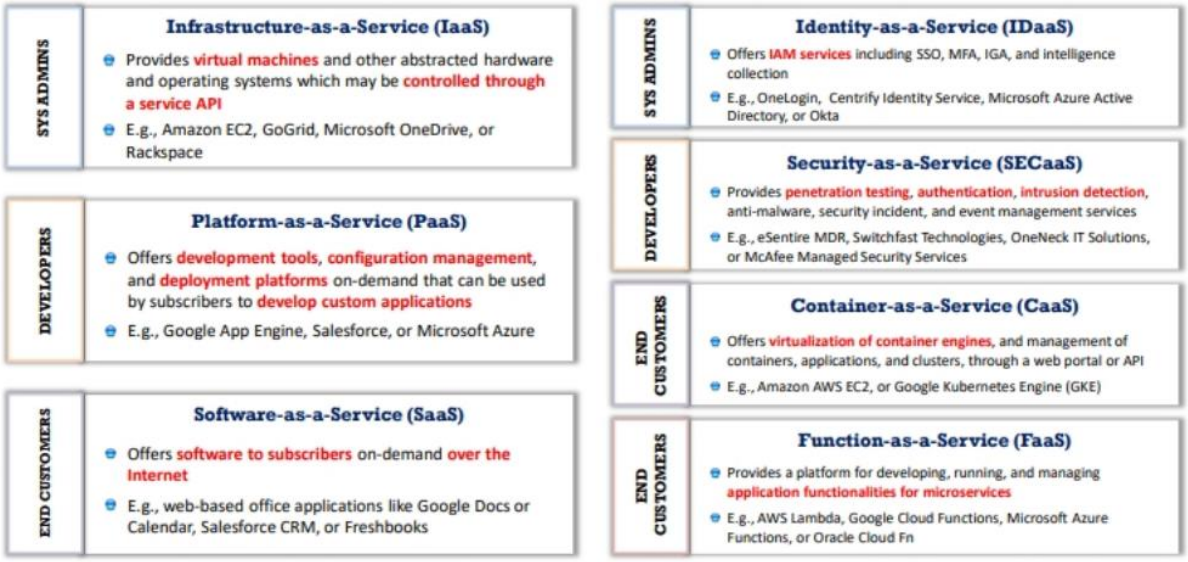
### Major Characteristics

- On-demand self-service
- Distributed Storage
- Rapid Elasticity
- Board network access
- Resource pooling
- Measured Services
- Automated management
- Virtualization

### Limitations

- Limited control and flexibility of organizations
- Proneness to outages and other technical issues
- Security, privacy, and compliance issues
- Contracts and lock-ins
- Dependence on network connections
- Potential vulnerability to attacks as every component is online
- Difficulty in migrating from one service provider to another

## Type of cloud computing services




### *Infrastructure-as-a-Service (IaaS)*

#### Advantage

- Dynamic infrastructure scaling
- Guaranteed uptime
- Automation of administrative tasks
- Elastic load balancing (elb)
- Policy-based services
- Global accessibility

#### Disadvantages

- Software security is at high risk (3rd party providers are more prone to attacks)
- Performance issues and slow connection speeds

### *Platform-as-a-Service (PaaS)*

#### Advantage

- Simplified deployment
- Prebuilt business functionality
- Lower security risk compared to IaaS
- Instant community
- Pay-per-use model
- scalability

#### Disadvantages

- Vendor lock-in
- Data privacy

- Integration with the rest of the system applications

### *Software-as-a-Service (SaaS)*

#### Advantage

- Low cost
- Easy administration
- Global accessibility
- High compatibility (no specialized hardware or software is required)

#### Disadvantages

- Security and latency issues
- Total dependency on the internet
- Switching between SaaS vendor is difficult

### *Identity-as-a-Service (IDaaS)*

#### Advantage

- Low cost
- Improved security
- Simplify compliance
- Reduced time
- Central management of user accounts

#### Disadvantages

- Single server failure may disrupt the service or create redundancy on other authentication servers
- Vulnerable to account hijacking attacks

### *Security-as-a-Service (SecaaS)*

#### Advantage

- Low cost
- Reduced complexity
- Continuous protection
- Improved security through best security expertise
- Latest and updated security tools
- Rapid user provisioning
- Greater agility
- Increased time on core competencies

#### Disadvantages

- Increased attack surfaces and vulnerabilities

- Unknown risk profile
- Insecure API
- No customization to business needs
- Vulnerable to account hijacking attacks

### *Container-as-a-Service (CaaS)*

#### Advantage

- Streamlined development of containerized applications
- Pay-per-resource
- Increased quality
- Portable and reliable application development
- Low cost
- Few resources
- Crash of application container does not affect other containers
- Improved security
- Improved patch management
- Improved response to bugs
- High scalability
- Streamlined development

#### Disadvantages

- High operational overhead
- Platform deployment is the developer's responsibility

### *Function-as-a-Service (FaaS)*

#### Advantage

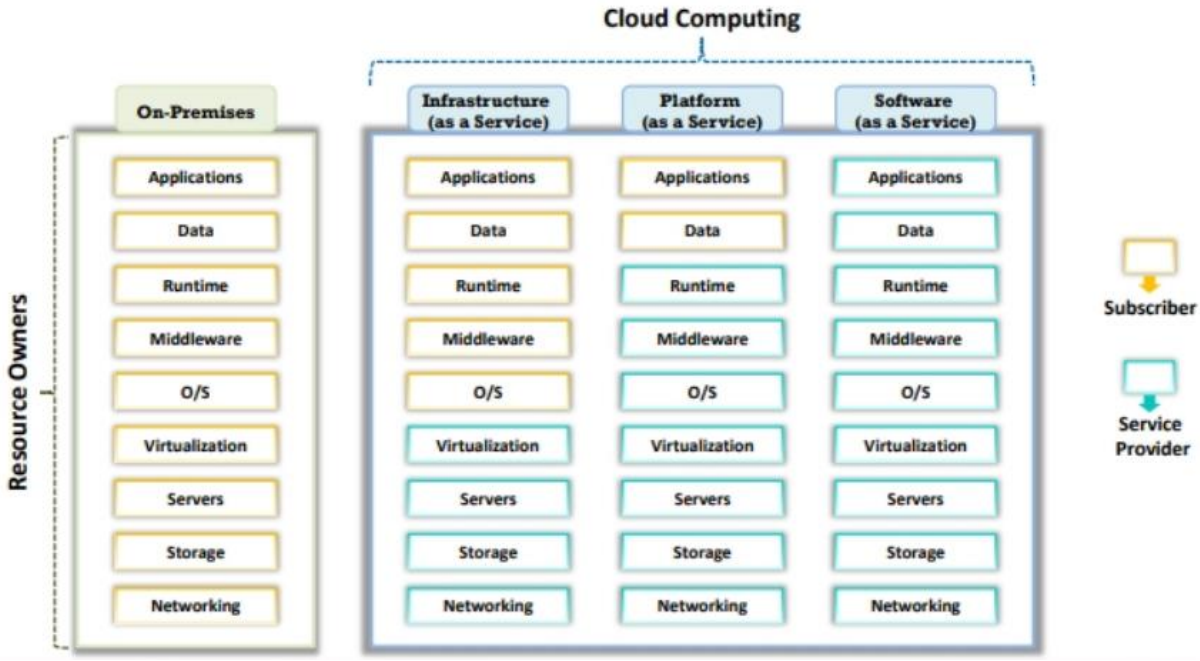
- Pay-per-use
- Low cost
- Efficient security updates
- Easy deployment
- High scalability

#### Disadvantages

- High latency
- Memory limitations
- Monitoring and debugging limitations
- Unstable tools and frameworks

- Vendor lock-in

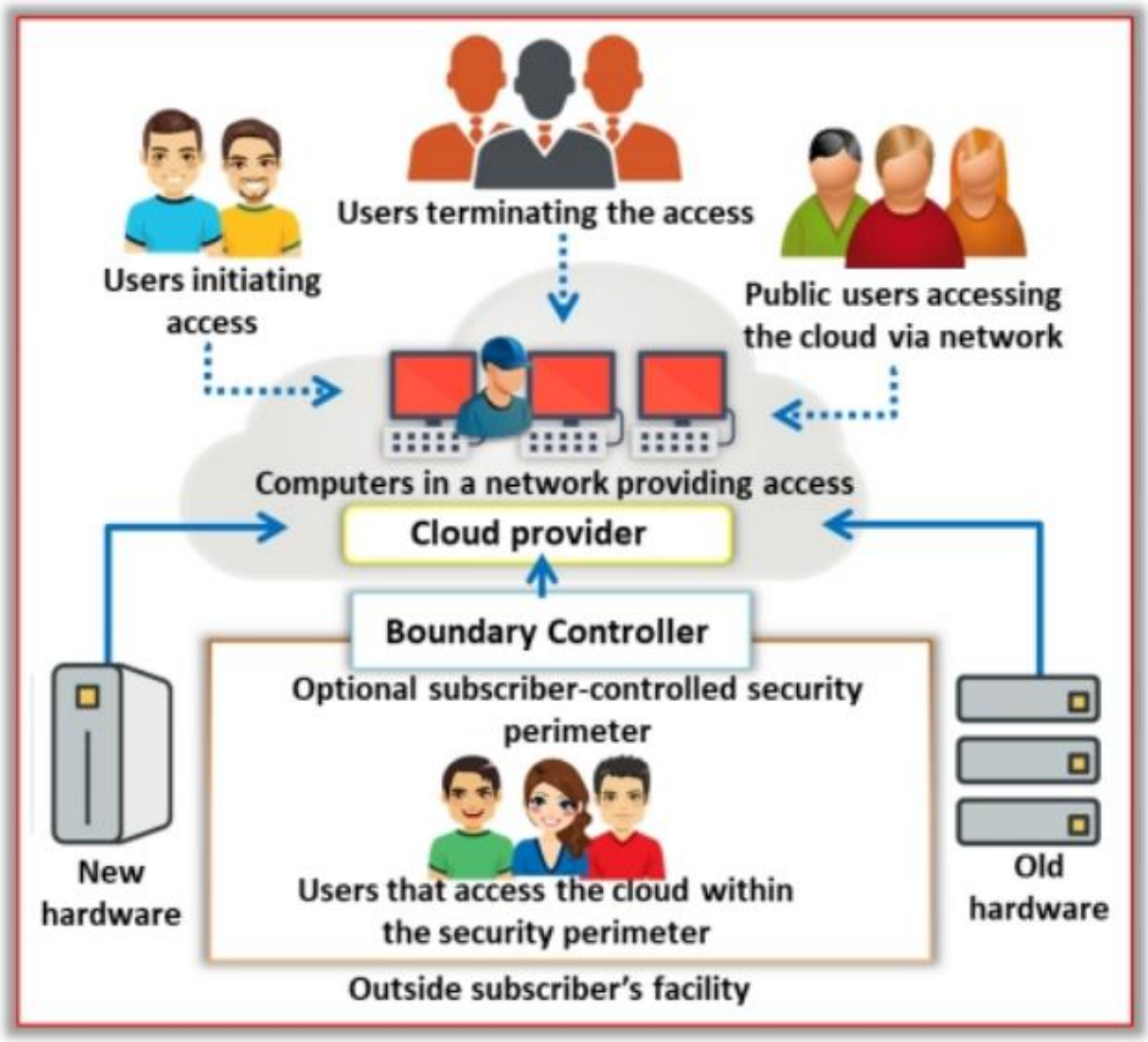
## Separation of responsible in cloud




## Cloud Deployment Models

### *Public cloud*

Services are rendered over network that is open for public use




### Advantage

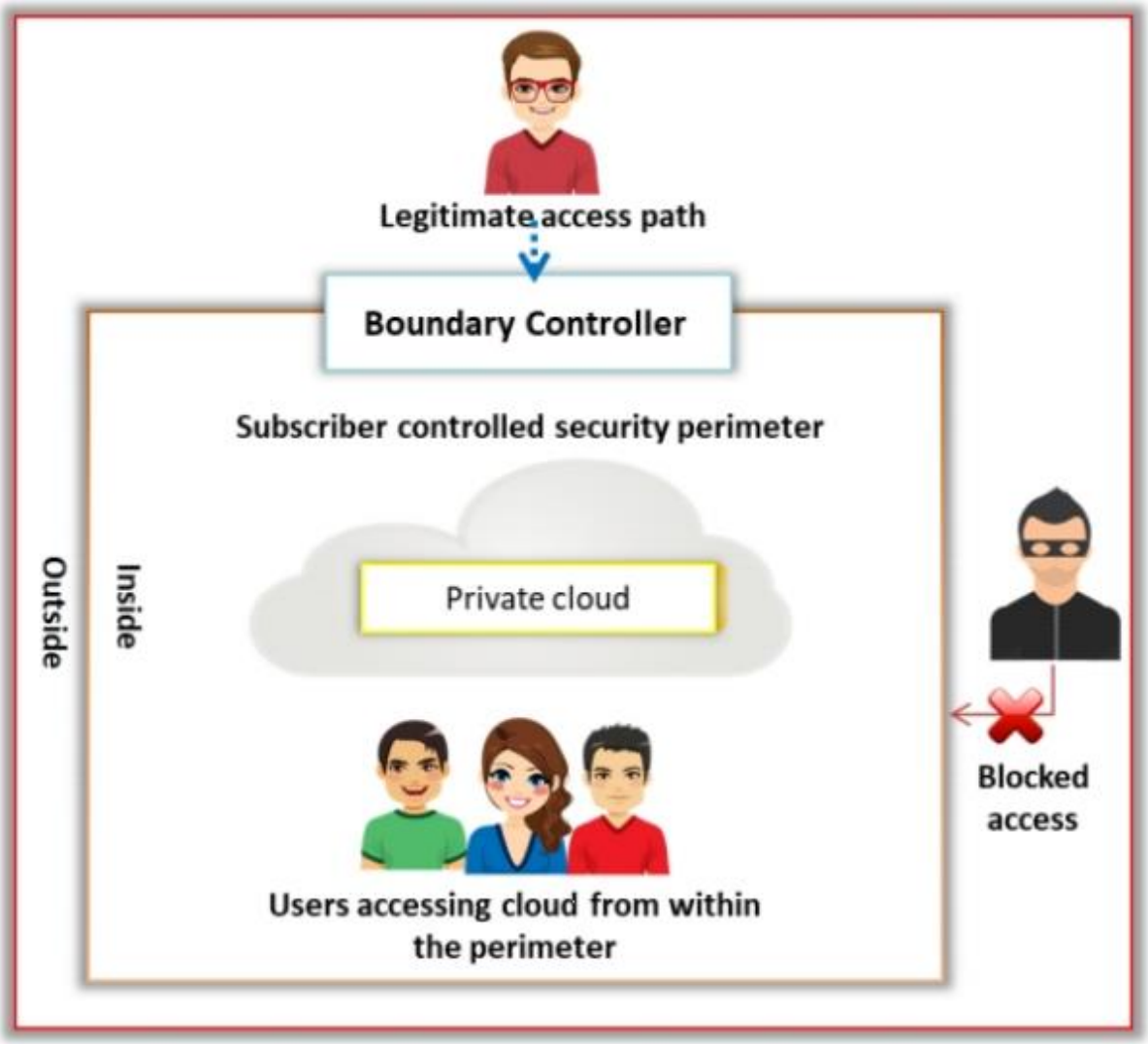
- Simplicity and efficiency
- Low cost
- Reduced time (when server crashes, needs to restart or reconfigure cloud)
- No maintenance (public cloud service is hosted off-site)
- No contracts (no long-term commitments)

### Disadvantages

- Security is not guaranteed
- Lack of control (third-party provides are in charge)
- Slow speed (relies on internet connections, the data transfer rate is limited)

## *Private cloud*

Cloud infrastructure is operated for a single organization only




### Advantage

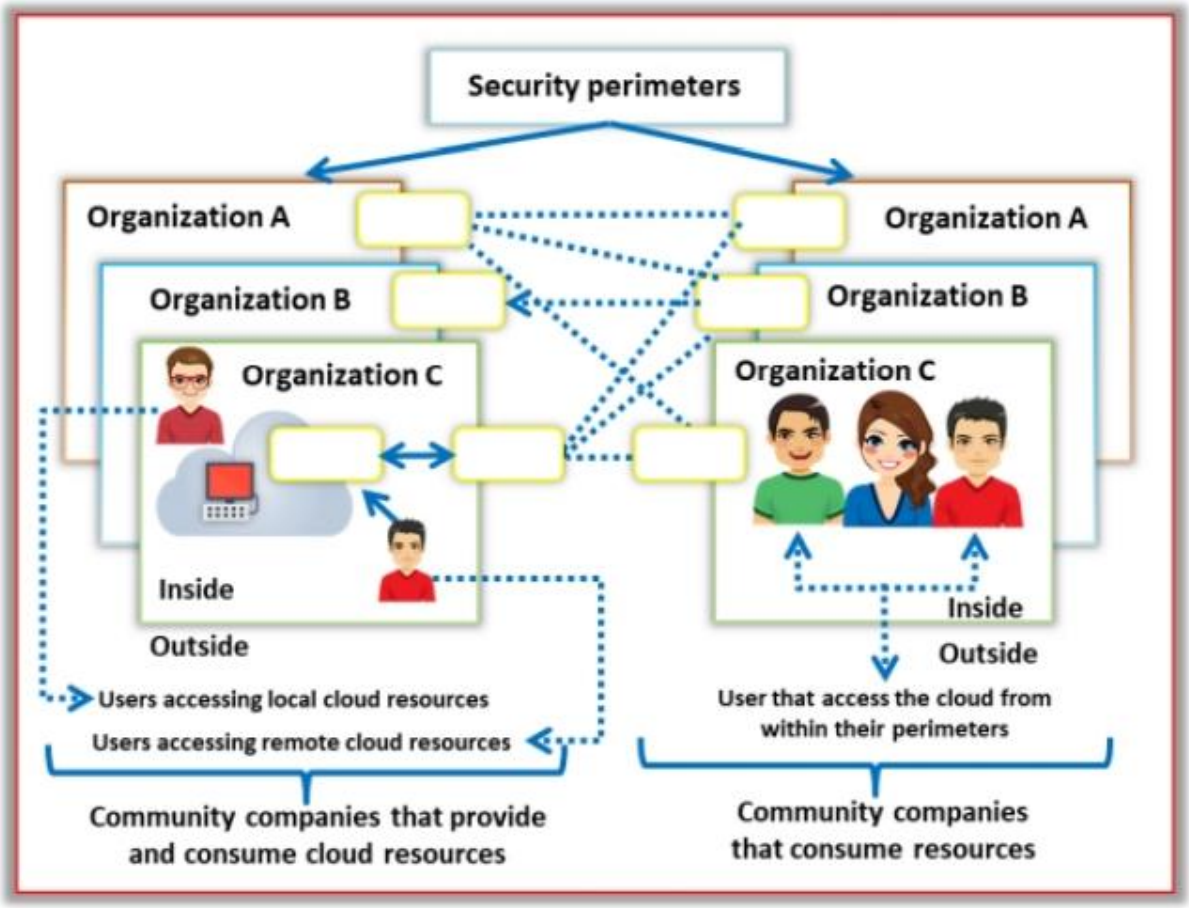
- Security enhancement (services are dedicated to a single organization)
- Increased control over resources (organization is in charge)
- High performance (cloud deployment within the firewall implies high data transfer rates)
- Customizable hardware, network and storage performances (as the organization owns private cloud)
- Sarbanes Oxley, PCI DSS, and HIPAA compliance data are much easier to attain

### Disadvantages

- High cost
- On-site maintenance

## *Community cloud*

Shared infrastructure between several organizations from a specific community with common concerns (Security, Compliance, Jurisdiction, etc.)




### Advantage

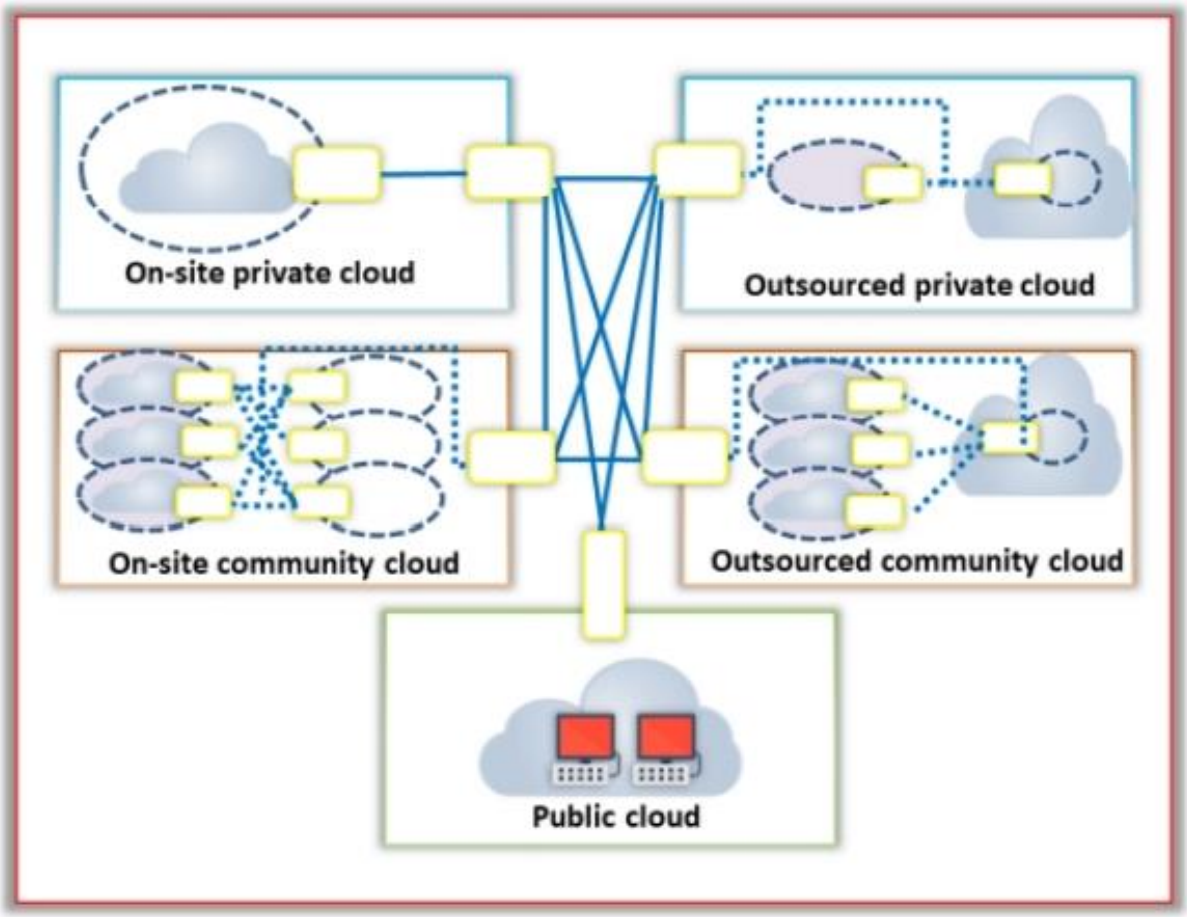
- Less expensive compared to private cloud
- Flexibility to meet the community's needs
- Compliance with legal regulations
- High scalability
- Organizations can share a pool of resources from anywhere via the internet

### Disadvantages

- Competition between consumers in resource usage
- Inaccurate prediction of required resources
- Lack of legal entity in case of liability
- Moderate security (other tenants may be able to access data)
- Trust and security concerns between tenants

## *Hybrid cloud*

Combination of two or more clouds (private, community, or public) that remain unique entities but are bound together, thereby offering the benefits of multiple deployment models




### Advantage

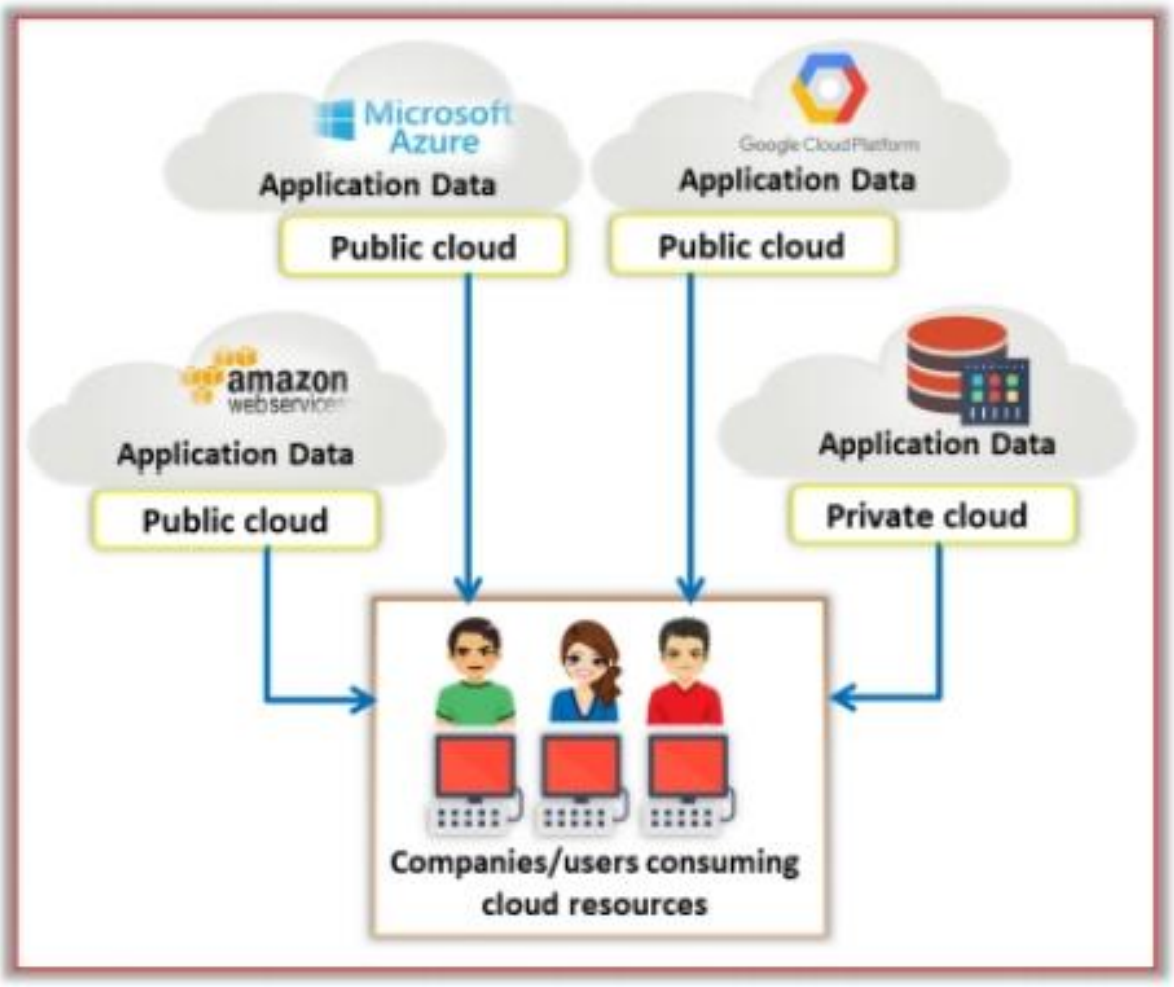
- High scalability (contains both public and private clouds)
- Offers both secure and scalable public resources
- High level of security (comprises private cloud)
- Allow to reduce and manage the cost according to requirements

### Disadvantages

- Communication at the network level may be conflicted as it uses both public and private clouds
- Difficult to achieve data compliance
- Organization reliant on the internal IT infrastructure in case of outages (maintain redundancy across data centers to overcome)
- Complex service level agreements (SLAs)

## *Multi cloud*

Dynamic heterogeneous environment that combines workloads across multiple cloud vendors, managed via one proprietary interface to achieve long term business goals




### Advantage

- High reliability and low latency
- Flexibility to meet business needs
- Cost-performance optimization and risk mitigation
- Low risk of distributed denial-of-service (DDoS) attacks
- Increased storage availability and computing power
- Low probability of vendor lock-in

### Disadvantages

- Multi-cloud system failure affects business agility
- Using more than one provider causes redundancy
- Security risk due to complex and large attack surface
- Operational overhead

## NIST Cloud computing reference architecture

NIST cloud computing reference architecture defines five major actors:

**Cloud Consumer**

A person or organization that uses cloud computing services

**Cloud Provider**

A person or organization providing services to interested parties

**Cloud Carrier**

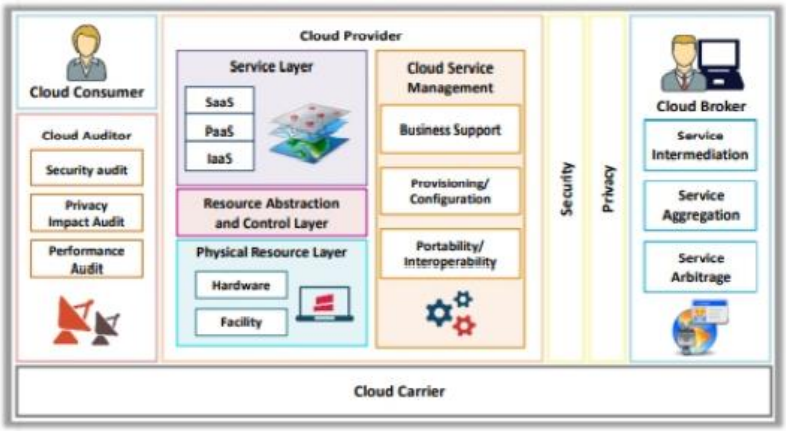
An intermediary for providing connectivity and transport services between cloud consumers and providers

**Cloud Auditor**

A party for making independent assessments of cloud service controls and taking an opinion thereon

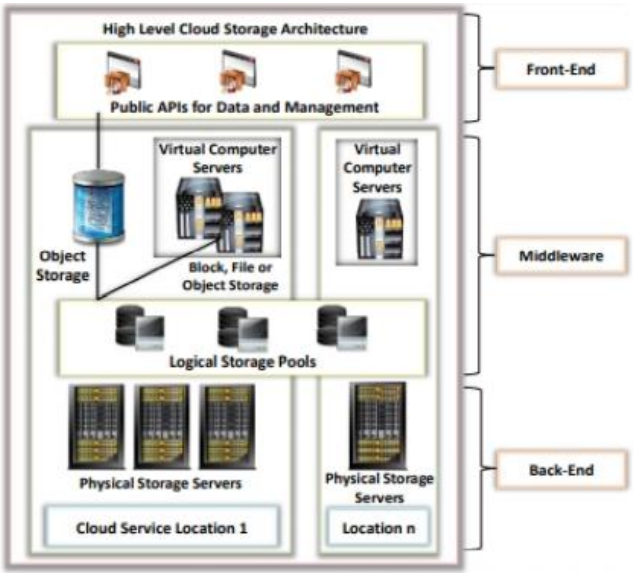
**Cloud Broker**

An entity that manages cloud services in terms of use, performance, and delivery, and maintains the relationship between cloud providers and consumers




## Cloud storage architecture

- Cloud storage is a data storage medium used to store digital data in logical pools using a network
- The cloud storage architecture consists of three main layers namely, front-end, middleware, and back-end
- The Front-end layer is accessed by the end user where it provides APIs for the management of data storage
- The Middleware layer performs several functions such as data de-duplication and replication of data
- The Back-end layer is where the hardware is implemented

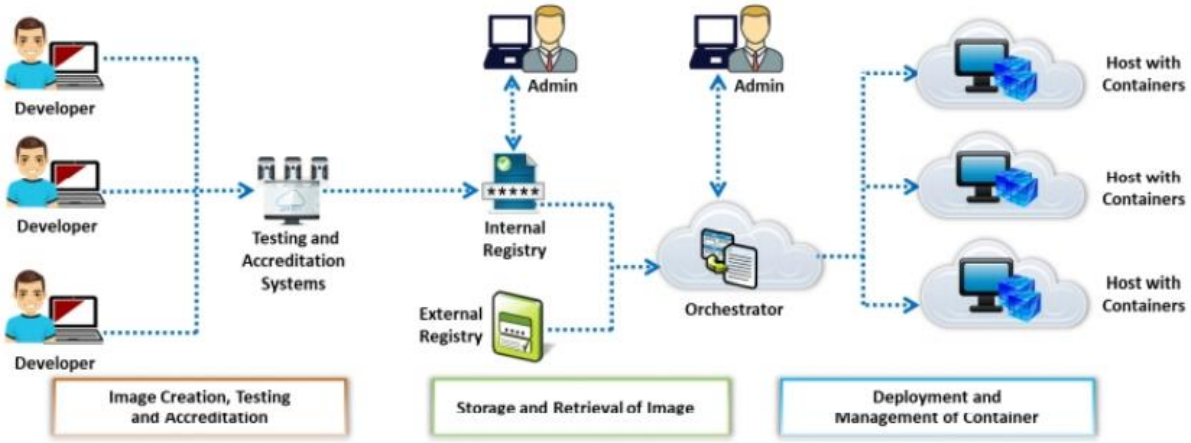



# Module Flow – Container Technology

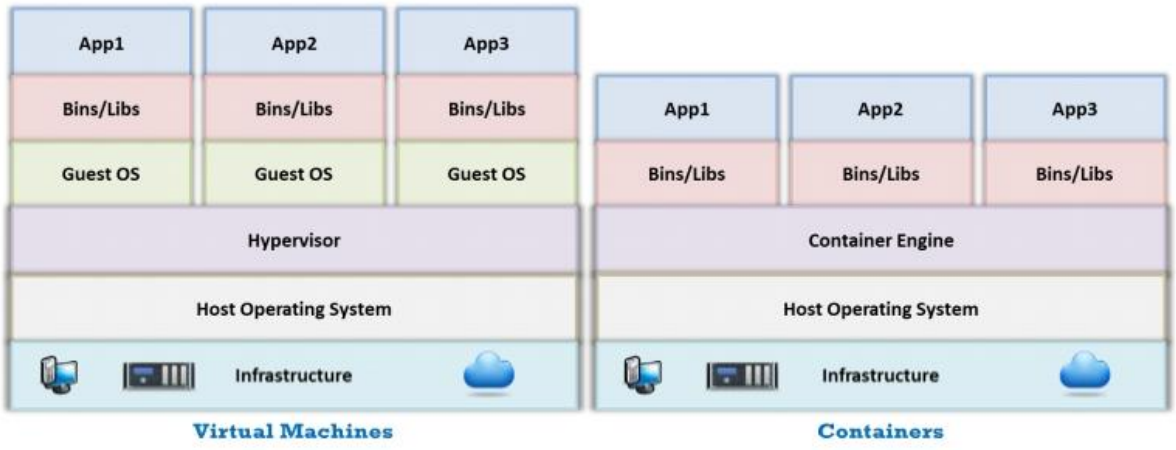
## What is a container?

- A container is a package of an application/software including all its dependencies such as library files, configuration files, binaries and other resources that run independently of other process in the cloud environment
- CaaS is a service that include the virtualization of containers and container management through orchestrators
- Using CaaS, subscribes can develop rich, scalable containerized application through the cloud or on-site data centers

## Architecture of container technology




## Container vs Virtual machine

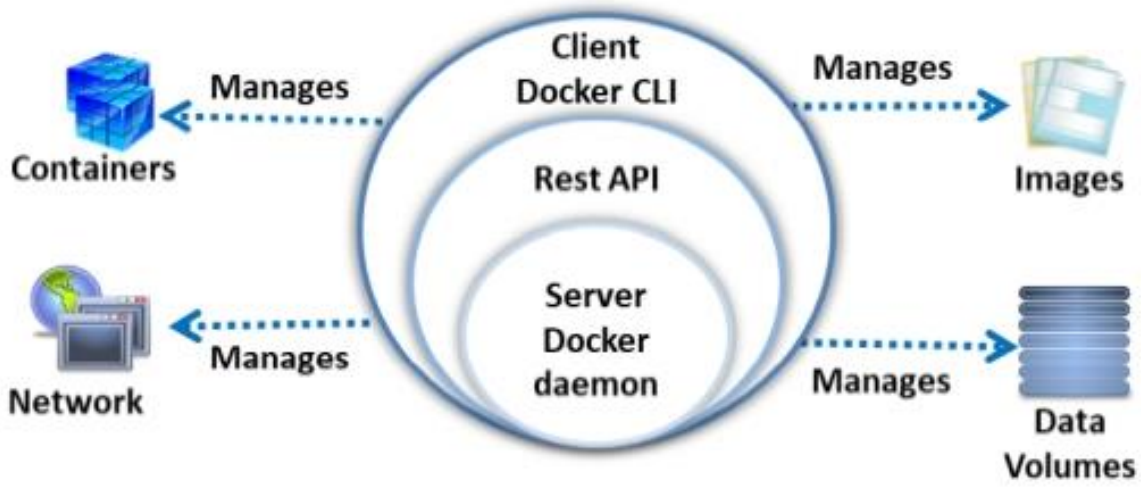


| Virtual Machines | Containers |
|---|---|
| Heavyweight | Lightweight and portable |
| Run on independent operating systems | Share a single host operating system |
| Hardware-based virtualization | OS-based virtualization |
| Slower provisioning | Scalable and real-time provisioning |
| Limited performance | Native Performance |
| Completely isolated making it more secure | Process-level isolation, partially secured |
| Created and lauched in minutes | Created and launched in second |

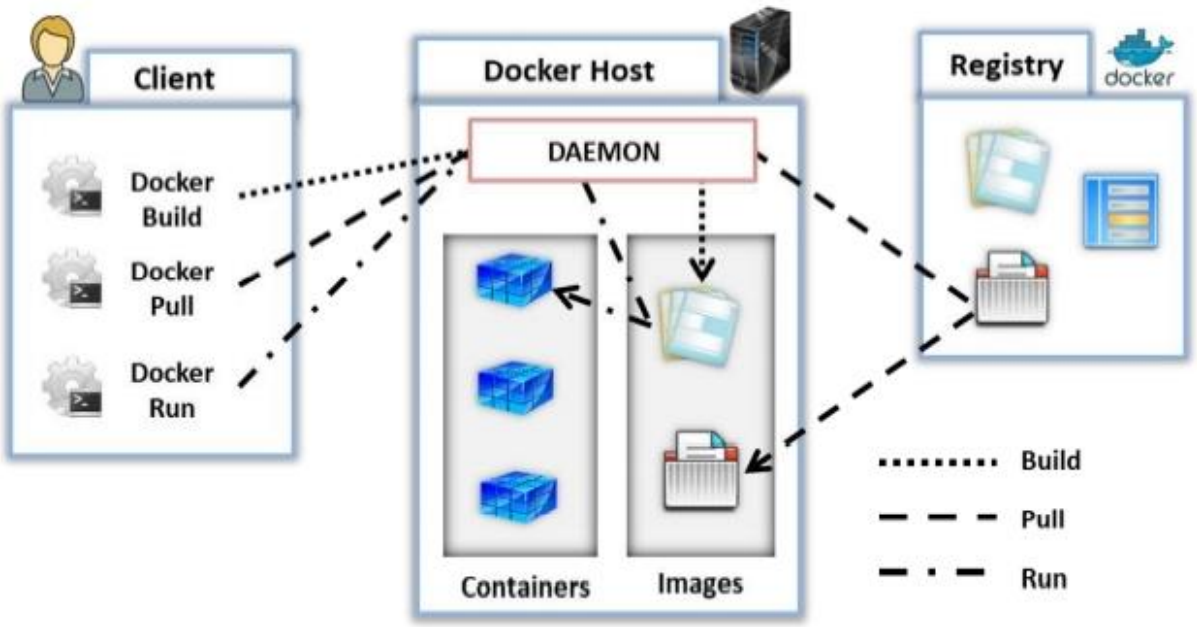
## What is docker ?

- Docker is an open source technology used for developing, packaging, and running applications and all its dependencies in the form of containers, to ensure that the application works in a seamless environment
- Docker provides a platform-as-service (PaaS) through os-level virtualization and delivers containerized software packages

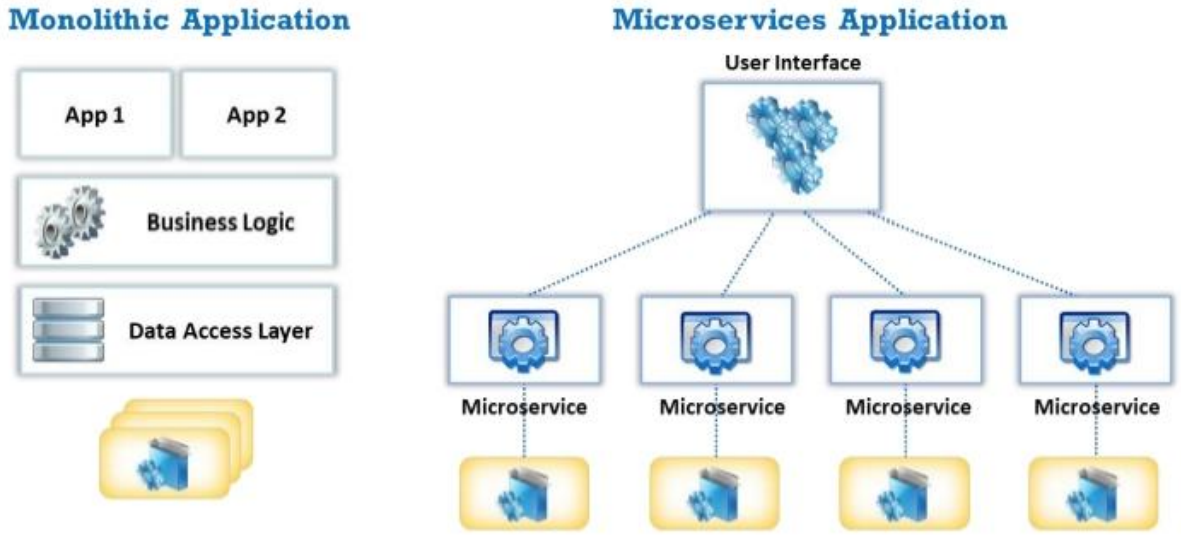
### Docker Engine



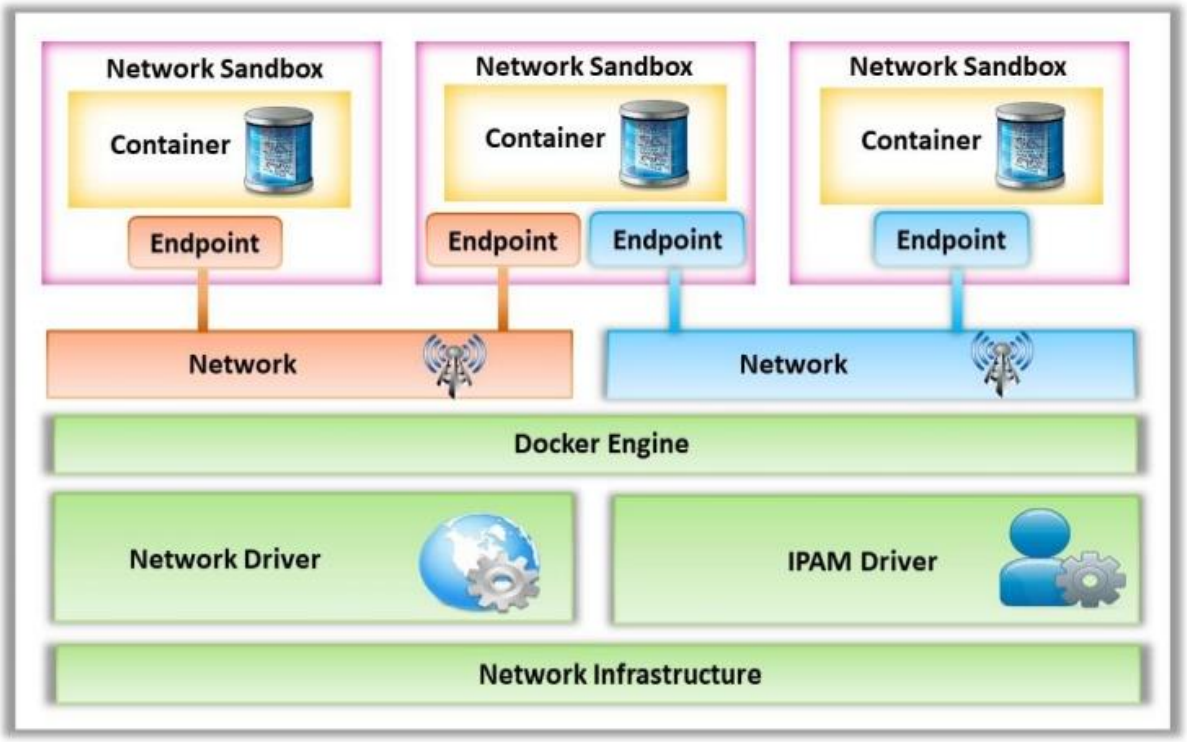
## Docker architecture




## Monolithic vs microservices application



## Container network model




## Docker native network drivers

**Host**: Allows the container to implement the host networking stack

**Bridge**: Creates a Linux bridge on the host that is managed by the Docker

**Overlay**: Enables container to container communication over the physical network infrastructure

**MACVLAN**: Creates a network connection between container interfaces and its parent host interface or sub-interfaces

**None**: Allows the container to implement its own networking stack and is isolated from the host networking stack

## What is Kubernetes ?

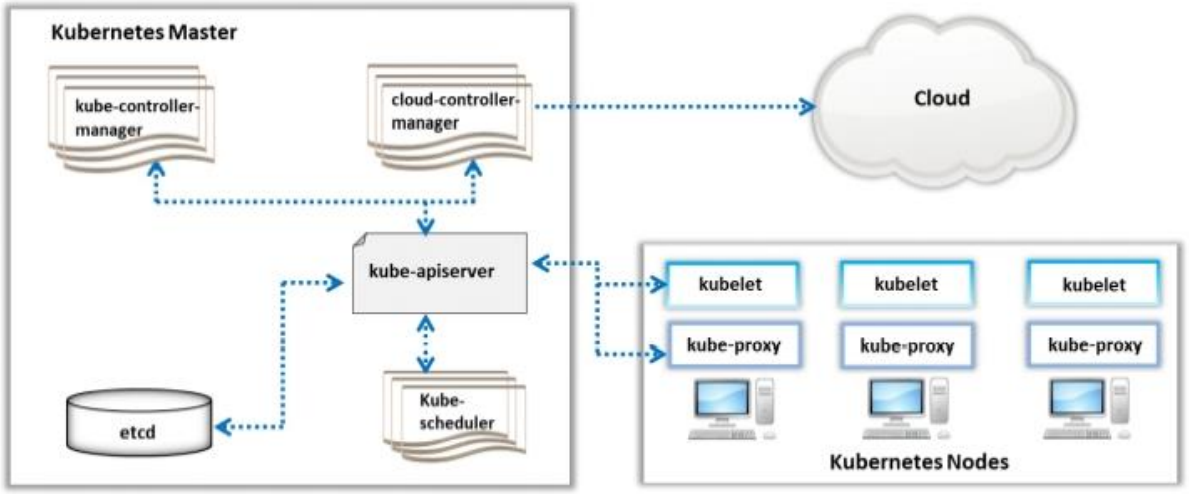
- Kubernetes, aka K8, is a open-source, portable, extensible, orchestration platform developed by google for managing containerized applications and microservices
- Kubernetes provides a resilient framework for managing distributed containers, generating deployment patterns, and performing failover and redundancy for the applications

### Features

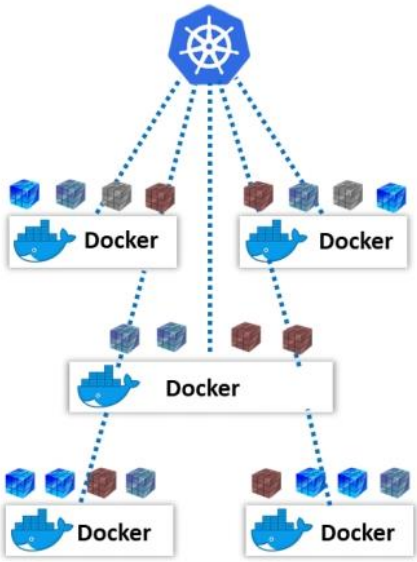
- Service discovery
- Load balancing
- Storage orchestration

- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management

## Kubernetes cluster architecture




## Kubernetes vs docker




Kubernetes and Docker run together to build and run containerized applications

• Docker is open source software that can be installed on any host to build, deploy, and run containerized applications on a single operating system

• When docker is installed on multiple hosts with different operating systems, you can use Kubernetes to manage these docker hosts

• Kubernetes is a container orchestration platform that automates the process of creating, managing, updating, scaling, and destroying containers

• Kubernetes can be coupled with any containerization technology as docker, rkt, runc, and cri-o

• Both docker and Kubernetes are based on micro services architecture, and built using the go programming language to deploy small, lightweight binaries, and YAML files for specifying application configurations and stacks

## Container security challenges

- Inflow of vulnerable source code
- Large attack surface

- Lack of visibility
- Compromising secrets
- DevOps speed
- Noisy neighboring containers
- Container breakout to the host
- Network-based attack
- Bypassing isolation
- Ecosystem complexity

## Module Flow – Serverless Computing


### What is serverless computing ?

- Serverless computing aka serverless architecture or function-as-a-service (FaaS), is a cloud-based application architecture where application infrastructure and supporting service are provided by the cloud vendor as they are needed
- Serverless computing simplifies the process of application deployment and eliminates the need for managing the server and hardware by the developers

### Advantage

- High scalability and flexibility
- Faster deployment and updating
- Reduced infrastructure cost
- No server management
- Pay-per-use
- Reduced latency and scaling cost
- Quicker provisioning of resources

- Low risk of failure
- No system administration

## Disadvantages

- Increased security vulnerability
- Vendor-lock-in
- Difficulty in managing statelessness
- Complex end-to-end application testing
- Unsuitability of long-running processes for serverless computing

## Serverless architecture


## Containers vs serverless computing

| Containers | Serverless computing |
|---|---|
| The developer is responsible for defining container configuration files along with the operating system, software, libraries, storage, and networking | The developer only needs to develop and upload code to support serverless computing, the entire process is taken care of by the cloud service provider |
| Once initiated, the container runs continuously until the developer stops or destroys it | Once completed running, the serverless is automatically destroyed by the cloud environment |
| A container need server support even when the container is not executing any programs | Serverless deployment charges only for the resources consumed |

| | |
|---|---|
| There is no time restriction for code running inside the container | Timeout is enabled on serverless functions |
| Containers support running on a cluster of hosts nodes | The underlying host infrastructure is transparent to developers |
| Containers store data in temporary storage or mapped storage volumes | Serverless functions do not support temporary storage; instead, data is stored in the object storage medium |
| Containers support both complex applications and lightweight microservices | Serverless functions are suitable only for microservices applications |
| Developers can select their choice of language and runtime for applications running in a containers | Language selection for serverless function is restricted by the cloud service provider |

## Module Flow – Cloud Computing Threats


### OWASP Top 10 Cloud Security Risks

R1 – Accountability & Data Ownership

Using the public cloud for hosting business services can cause severe risk for the recoverability of data

R2 – User Identity Federation

Creating multiple user identities for different cloud providers makes it complex to manage multiple user IDs and credentials

R3 – Regulatory Compliance

There is a lack of transparency, and there are different regulatory laws in different countries

R4 - Business Continuity & Resiliency

There can be business risk or monetary loss if the cloud provider handles the business continuity improperly

R5 – User Privacy & Secondary Usage of Data

The default share feature in social web sites can jeopardize the privacy of user's personal data

R6 - Service & Data Integration

Unsecured data in transit is susceptible to eavesdropping and interception attacks

R7 - Multi-tenancy & Physical Security

Poor logical segregation may lead to tenants interfering with the security features of other tenants

R8 - Incidence Analysis & Forensics Support

Due to the distributed storage of logs across the cloud, law enforcement agencies may face problems in forensics recovery

R9 - Infrastructure Security

Misconfiguration of infrastructure may allow network scanning for vulnerable applications and services

R10 - Non-production Environment Exposure

Using non-production environments increases the risk of unauthorized access, information disclosure, and information modification

## OWASP Top 10 Serverless Security Risks

| Risks | Security Weakness | Impact |
|---|---|---|
| Al - Injection | SQL/NOSQL injection, OS command injection, Code injection | The impact depends on the permissions of the vulnerable function |
| A2 - Broken Authentication | Poor design of identity and access controls | Leads to sensitive data leakage, breaking of the system's business logic, and execution flow |
| A3 - Sensitive Data Exposure | Storing sensitive data in plaintext or using weak encryption | Exposure of sensitive data such as PII, health records, credentials, and credit card details |
| A4 - XML External Entities (XXE) | Using XML processors might make the application vulnerable to XXE attacks | Leakage of function code and sensitive files |

| | | |
|---|---|---|
| A5 - Broken Access Control | Granting functions access and more privileges to unnecessary resources | The impact depends on the compromised resource |
| A7 - Security Misconfiguration | Poor patch management, functions with long timeout and low concurrency | Sensitive information leakage, loss of money, DoS, and unauthorized access to cloud resources |
| A7 - Cross-Site Scripting (XSS) | Untrusted input used to generate data without properly escaping | User impersonation, access to sensitive data such as API keys |
| A8 - Insecure Deserialization | Deserialization vulnerabilities in Python, JavaScript, etc. | Running arbitrary code, data leakage, resource and account control |
| A9 - Using Components with Known Vulnerabilities | Lack of knowledge on component-heavy deployment patterns | The business impact depends on the specification of the known vulnerabilities |
| A10 - Insufficient Logging and Monitoring | Insufficient security monitoring and auditing | The impact of not identifying a security incident timeously can be significant |

## Cloud Computing Threats

| | | | | | |
|---|---|---|---|---|---|
| 1 | Data breach/loss | 13 | Loss of business reputation due to co-tenant activities | 26 | Licensing risks |
| 2 | Abuse and Nefarious Use of Cloud services | 14 | Privilege escalation | 27 | Loss of governance |
| 3 | Insecure interfaces and APIs | 15 | Natural disasters | 28 | Loss of encryption keys |
| 4 | Insufficient due diligence | 16 | Hardware failure | 29 | Risks from changes of Jurisdiction |
| 5 | Shared technology issues | 17 | Supply chain failure | 30 | Undertaking malicious probes or scans |
| 6 | Unknown risk profile | 18 | Modifying network traffic | 31 | Theft of computer equipment |
| 7 | Unsynchronized system clocks | 19 | Isolation failure | 32 | Cloud service termination or failure |
| 8 | Inadequate infrastructure design and planning | 20 | Cloud provider acquisition | 33 | Subpoena and e-discovery |
| 9 | Conflicts between client hardening procedures and cloud environment | 21 | Management interface compromise | 34 | Improper data handling and disposal |

| | | | | | |
|---|---|---|---|---|---|
| 10 | Loss of operational and security logs | 22 | Network management failure | 35 | Loss or modification of backup data |
| 11 | Malicious insiders | 23 | Authentication attacks | 36 | Compliance risks |
| 12 | Illegal access to cloud systems | 24 | VM-level attacks | 37 | Economic Denial of Sustainability (EDOS) |
| | | 25 | Lock-in | 38 | Lack of Security Architecture |
| | | | | 39 | Hijacking Accounts |

## Container Vulnerabilities

| | | | | | |
|---|---|---|---|---|---|
| 1 | Impetuous Image Creation | Careless creation of images by not considering the security safeguards or control aspects | 7 | Non-Updated Images | Outdated images may contain security loopholes and bugs that compromise the security of images |
| 2 | Unreliable Third-Party Resources | Untrusted third-party resources make the resources vulnerable to many malicious attacks | 8 | Hijacked Repository and Infected Resources | Security misconfiguration and bugs may allow attackers to gain unauthorized access to the repository so that they can poison the resources by altering or deleting files |
| 3 | Unauthorized Access | Gaining access to the user accounts leads to privilege escalation attacks | 9 | Hijacked Image Registry | Mismanaged configurations and vulnerabilities can be exploited to compromise the registry and image hubs |
| 4 | Insecure Container Runtime Configurations | Improper handling of the configuration option and mounting sensitive directories on the host can cause faulty and insecure runtime configurations | 10 | Exposed Services due to Open Ports | Misconfiguration of an application may allow open ports that expose sensitive information upon port scanning |
| 5 | Data Exposure in Docker Files | Docker images exposing sensitive information like passwords and SSH encryption keys can be exploited to compromise the security of the container | 11 | Exploited Applications | Vulnerable applications can be exploited using various techniques such as SQLi, XXS, and RFI |

| | | | | | |
|---|---|---|---|---|---|
| 6 | Embedded Malware | A container image may be embedded with malware after creation, or hardcoded functions may download malware after image deployment | 12 | Mixing of Workload Sensitivity Levels | Orchestrators place workloads having different sensitivity levels on the same host. One of the containers hosting a public webserver with vulnerabilities may pose a threat to the container processing sensitive information |

## Kubernetes Vulnerabilities

No Certificate Revocation

- Kubernetes does not support certificate revocation
- Attackers can exploit the certificate before it is replaced across the entire cluster

Unauthenticated HTTPS Connections

- Though Kubernetes uses PKI, connections between the components are not authenticated properly
- Attackers can gain unauthorized access to kubelet-managed Pods and retrieve sensitive information

Exposed Bearer Tokens in Logs

- Bearer tokens are logged in hyperkube kube-apiserver system logs
- Attackers having access to the system logs can impersonate a legitimate user

Exposure of Sensitive Data via Environment Variables

- Environmental variables allow settings to be derived from the variables
- Attackers can gain access to the stored values through environment logging

Secrets at Rest not Encrypted by Default

- Secrets defined by users are not encrypted by default
- Attackers gaining access to etcd servers can retrieve unencrypted secrets

Non-constant Time Password Comparison

- Kube-apiserver using basic password authentication, does not perform secure comparison of secret values
- Attackers can launch timing attacks to retrieve passwords

Hardcoded Credential Paths

- If the cluster token and the root CA are stored in different locations, an attacker can insert a malicious token and the root CA to gain access to the entire cluster

Log Rotation is not Atomic

- During log rotation, if the kubelet is restarted, all the logs may be erased
- The attacker waits for the log rotation to happen by monitoring it and then tries to remove all the logs

No Back-off Process for Scheduling

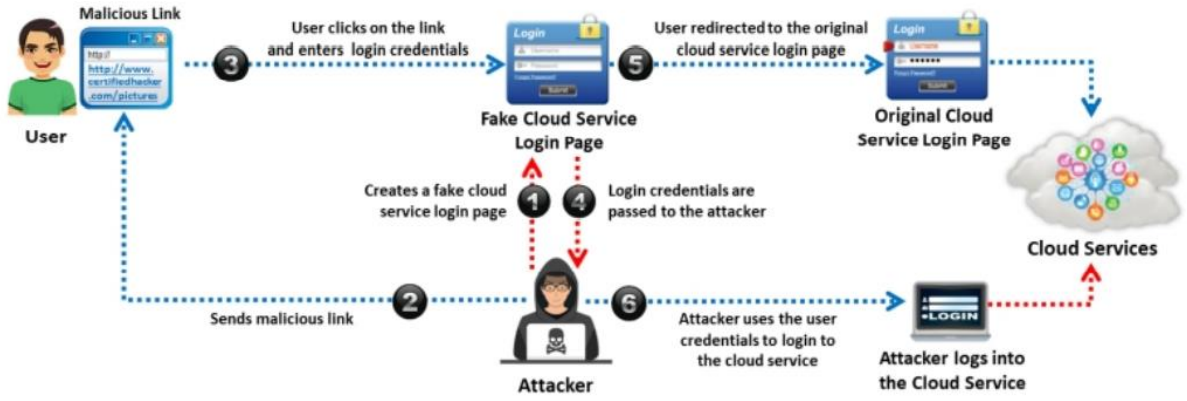
- No back-off process for scheduling the execution of Kubernetes pods
- This causes a tight loop as the scheduler continuously schedules a pod that is rejected by the other processes

No Non-repudiation

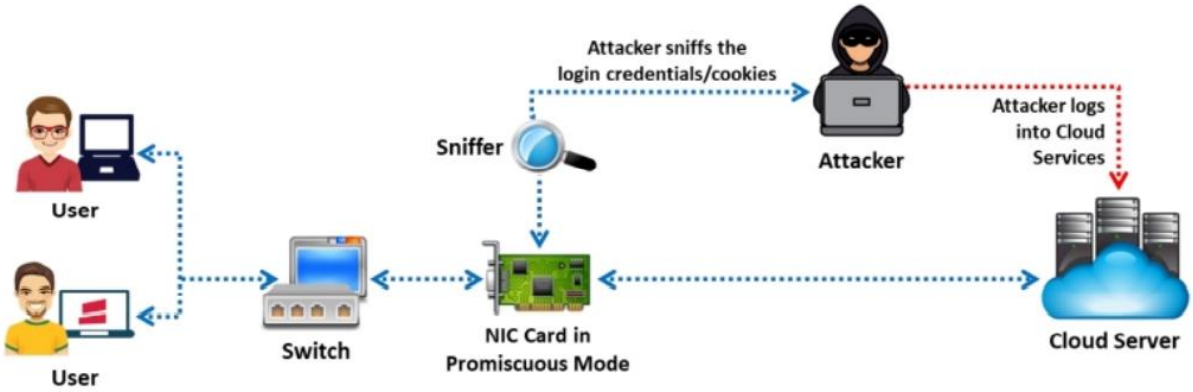
- If debug mode is disabled, kube-apiserver does not record user actions
- Attackers can directly interact with kube-apiserver and perform various malicious activities

## Cloud Attacks

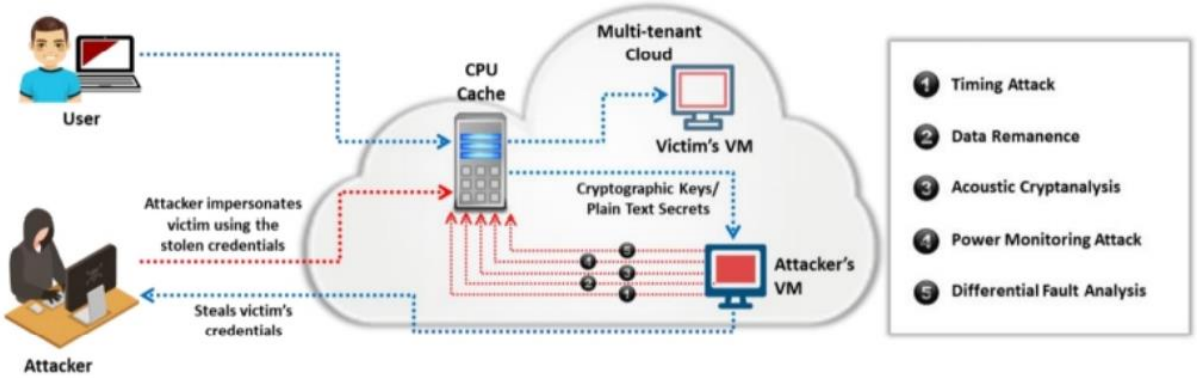
### Service Hijacking using Social Engineering



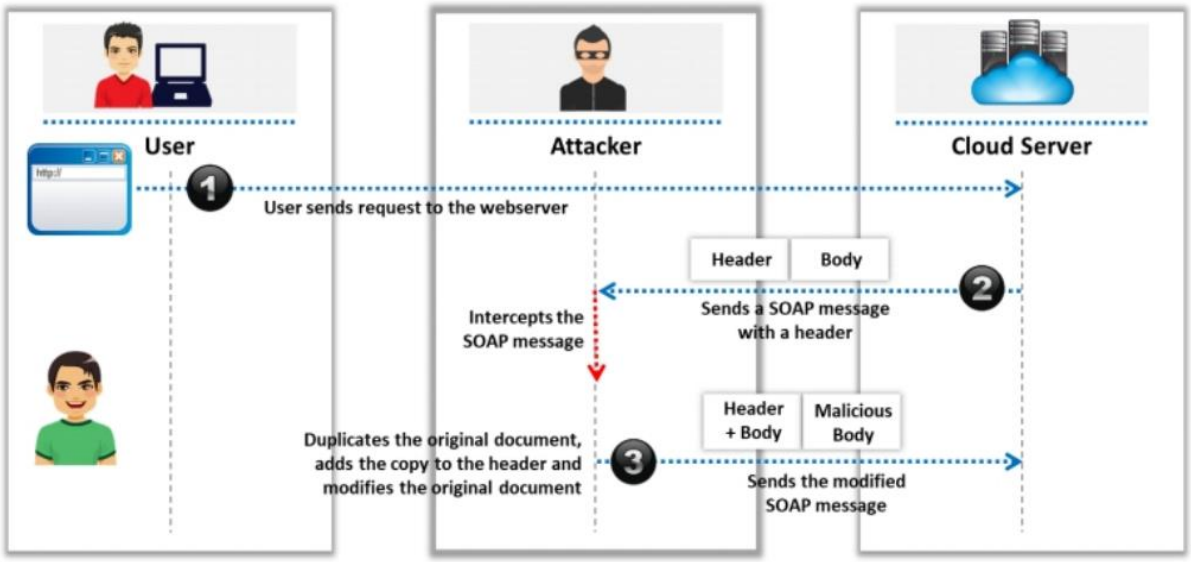

### Service Hijacking using Network Sniffing



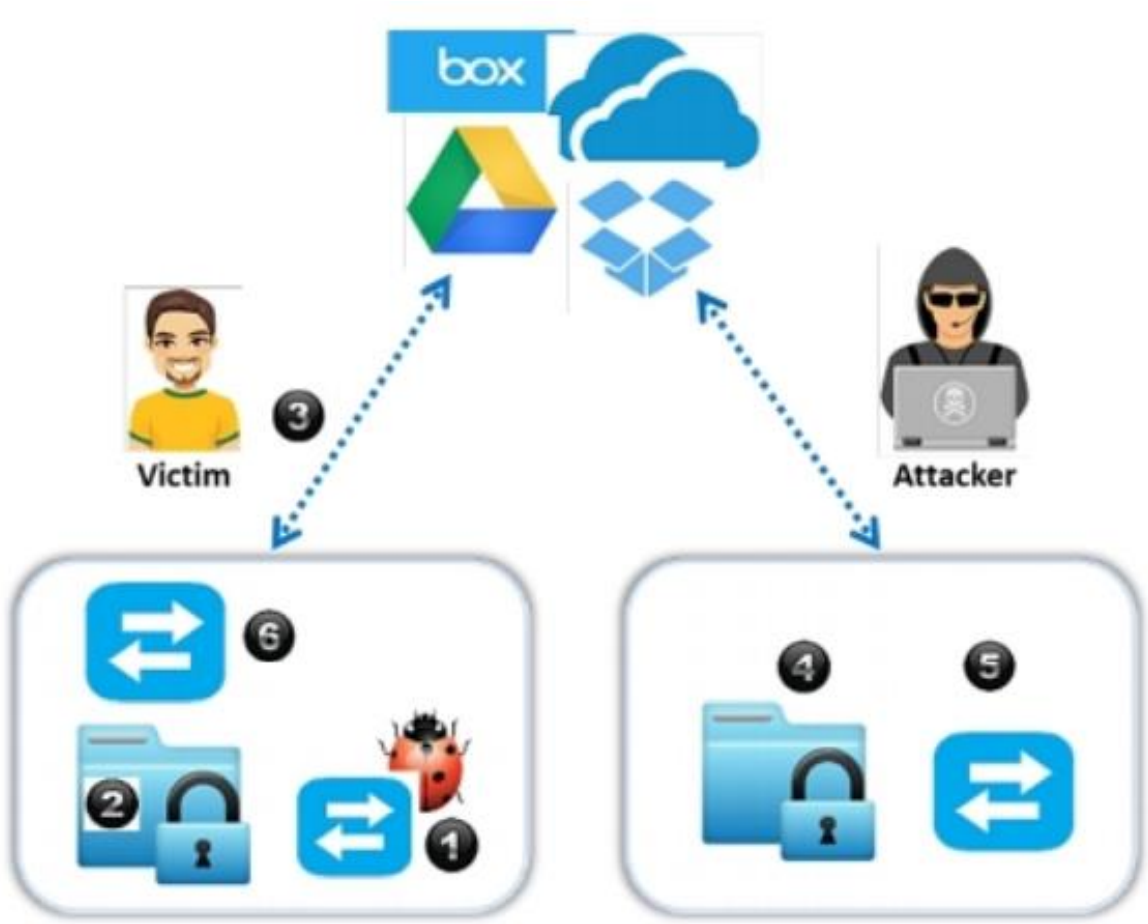
## Side-Channel Attacks or Cross-guest VM Breaches



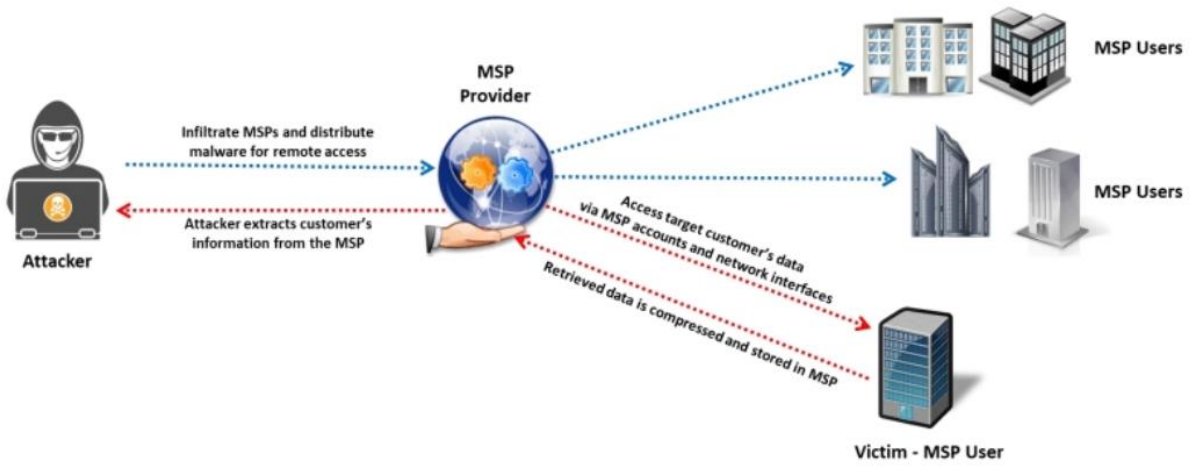

## Wrapping Attack



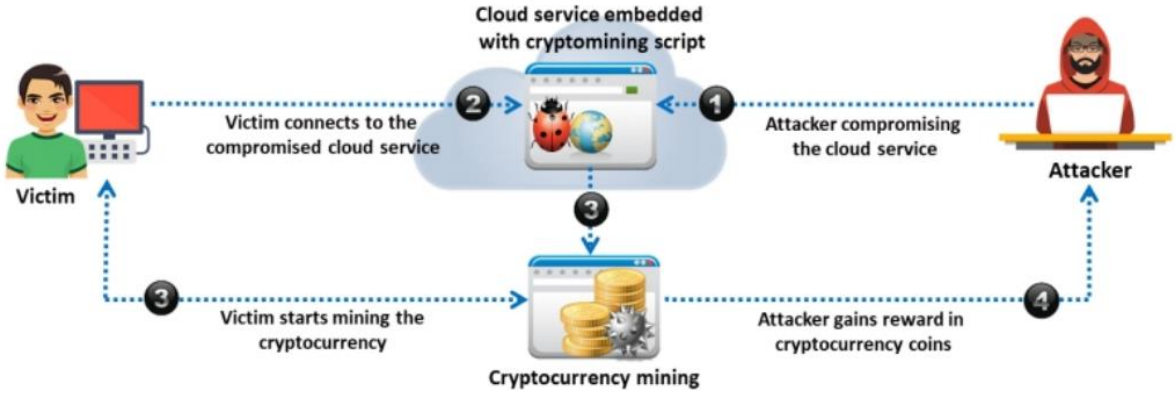
## Man-In-The-Cloud (MITC) Attack



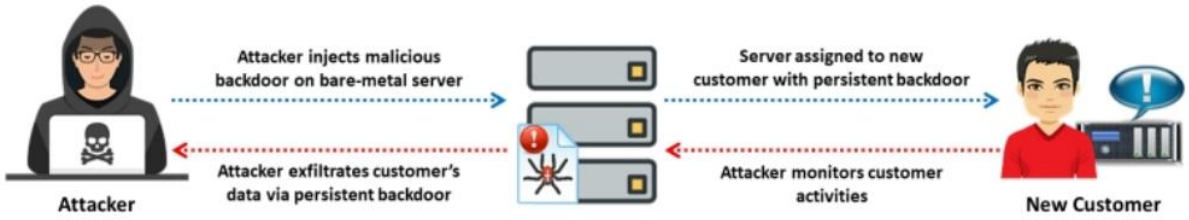

## Cloud Hopper Attack



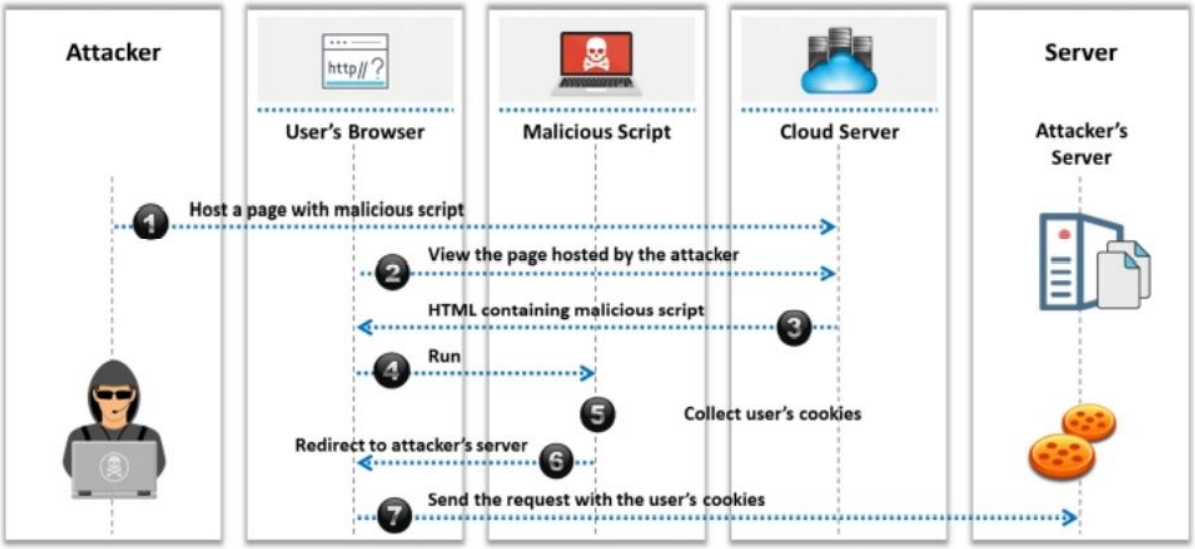
## Cloud Crypto jacking



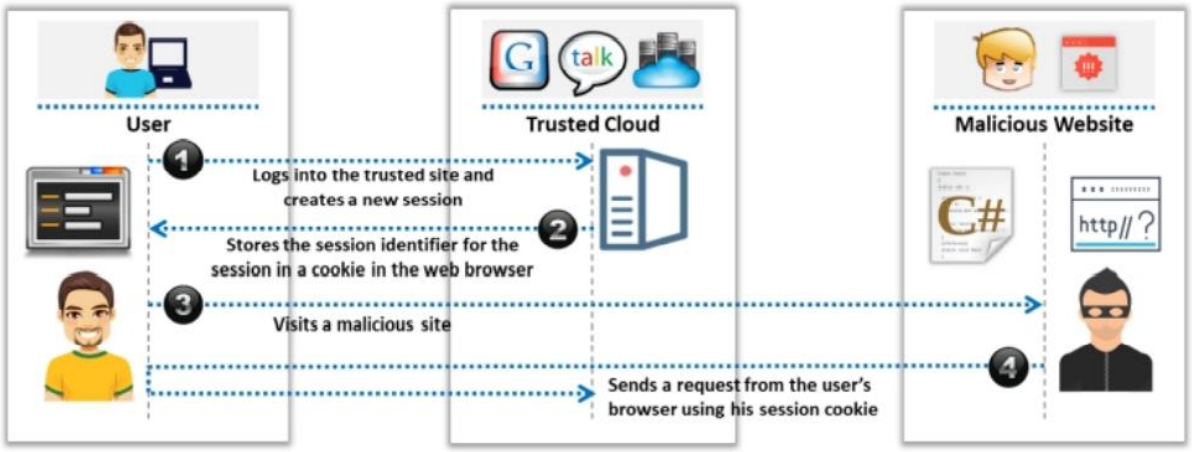

## Cloudborne Attack



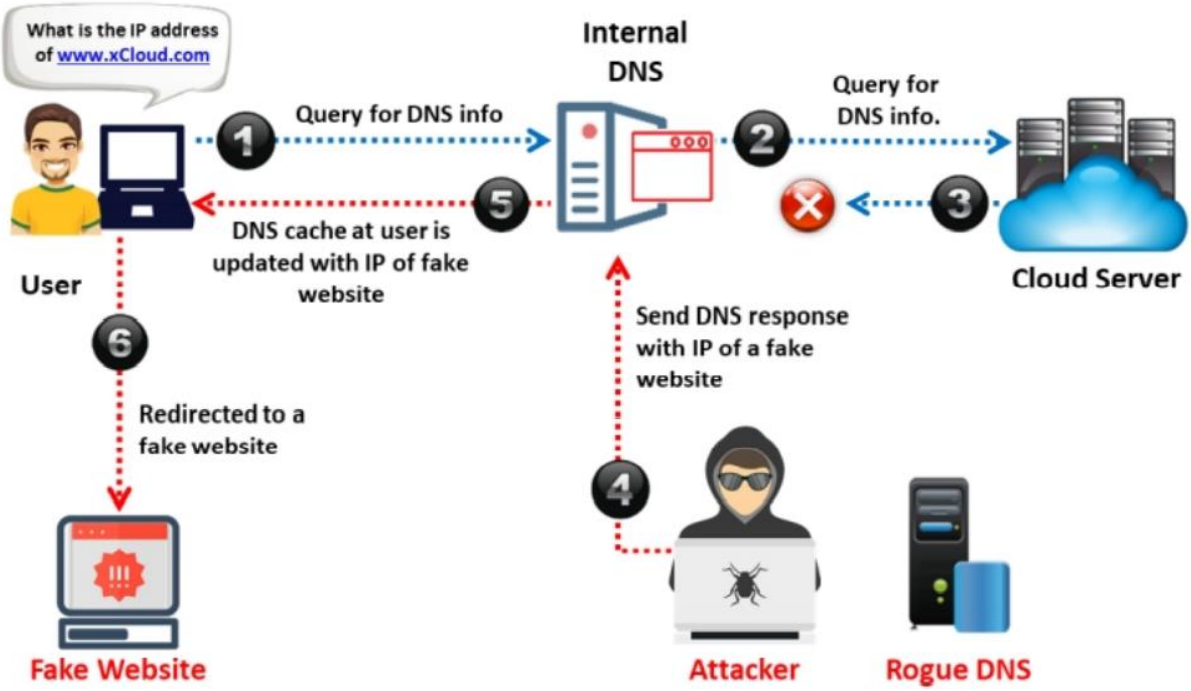

## Session Hijacking using Cross-Site Scripting (XSS) Attack



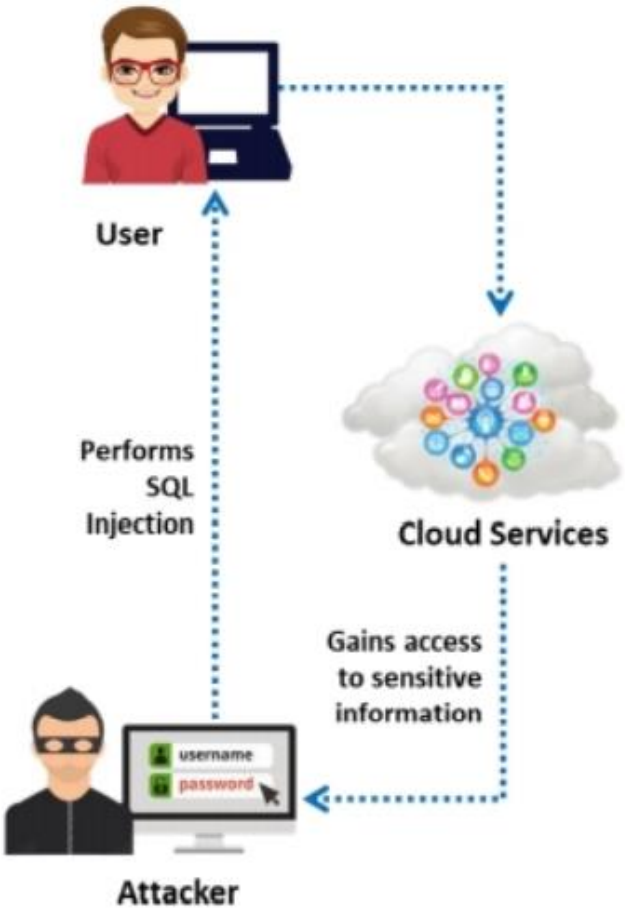
## Session Hijacking using Session Riding



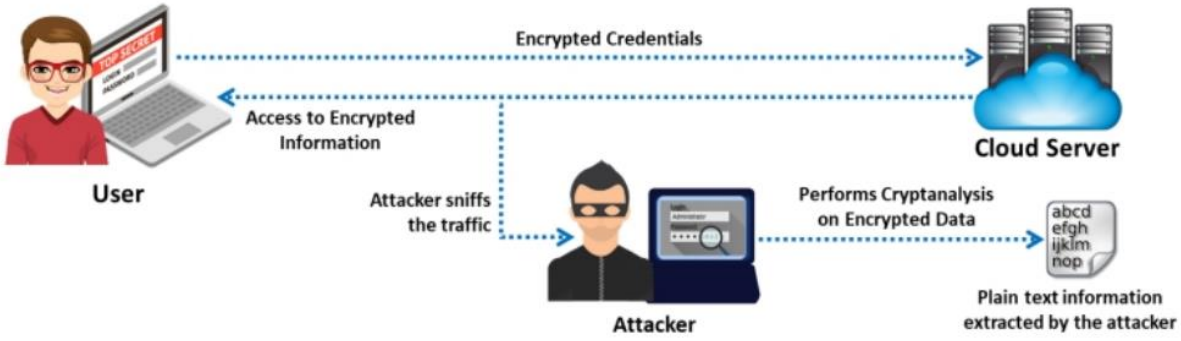

## Domain Name System (DNS) Attacks



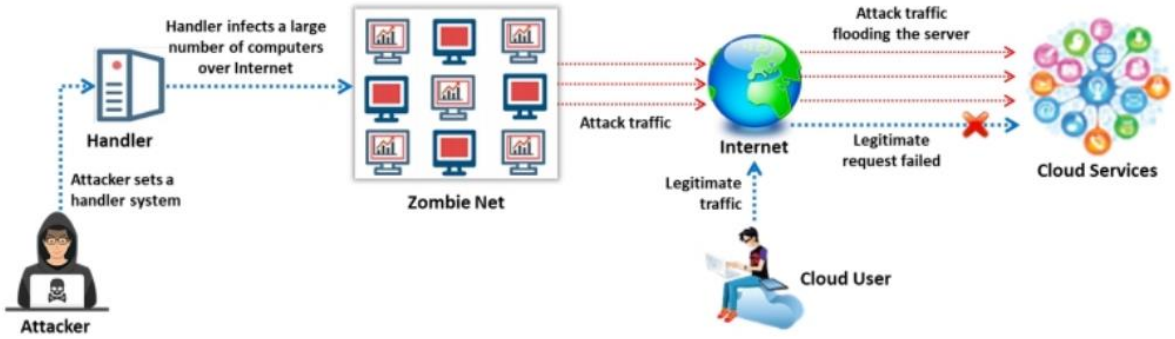
## Sql Injection Attacks



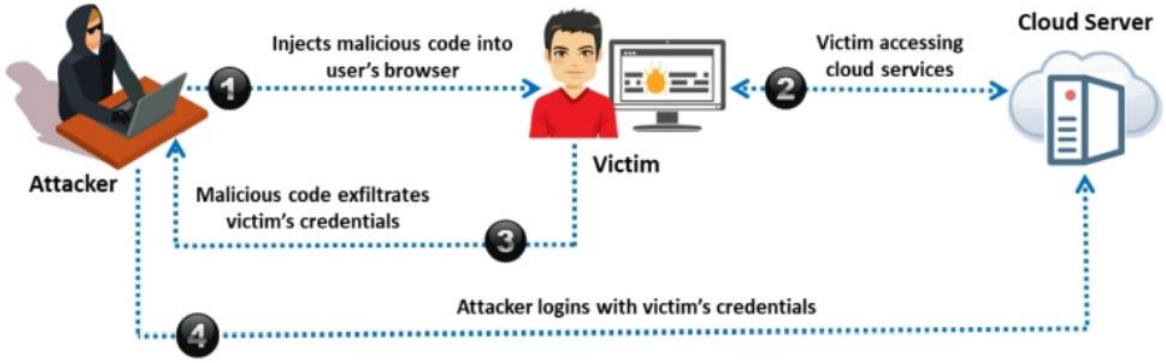

## Cryptanalysis Attacks



## DoS and DDoS Attacks




## Man-In-The-Browser Attack




# Module Flow – Cloud Hacking


## What is Cloud Hacking?

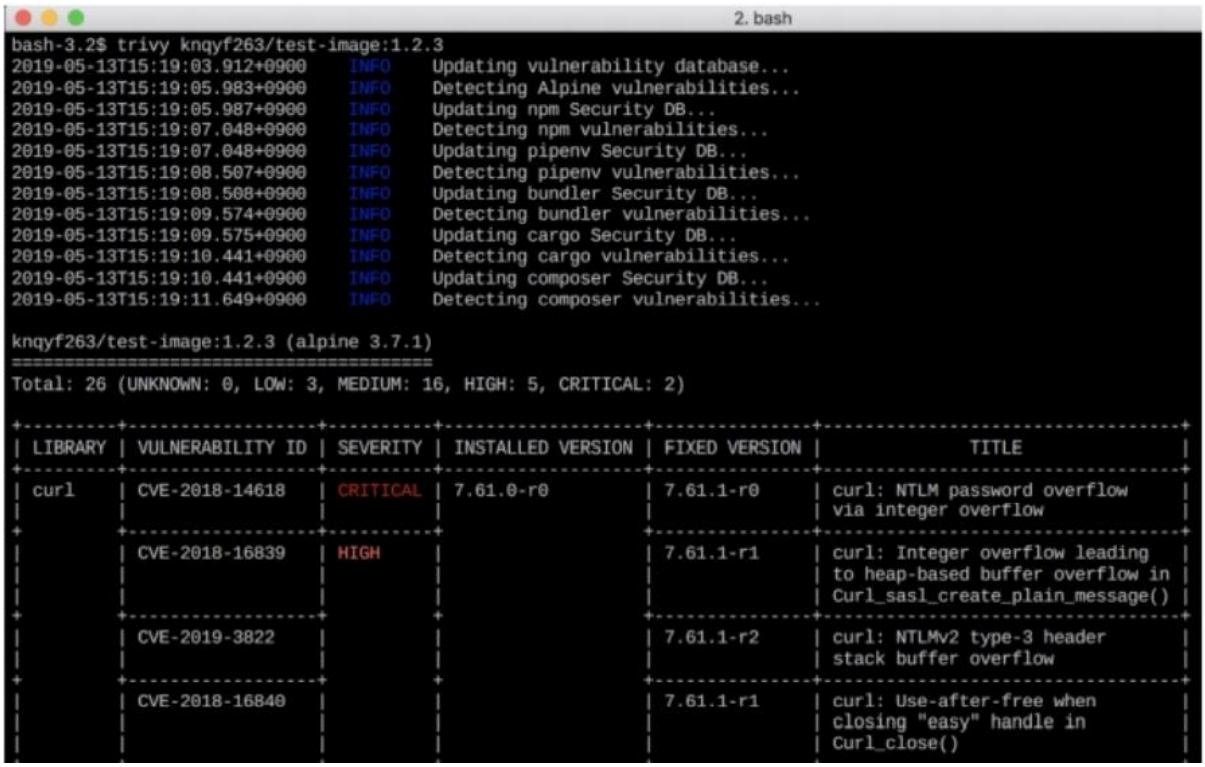
- Attackers exploit vulnerabilities existing in the cloud technologies to perform various targeted high-profile attacks on cloud storage systems, comprimising the corporate and customer's data
- The main objective of hacking the cloud enviroment is gaining access to user's data and blocking access to cloud services

## What Profit do Hacker got from Hacking Cloud?

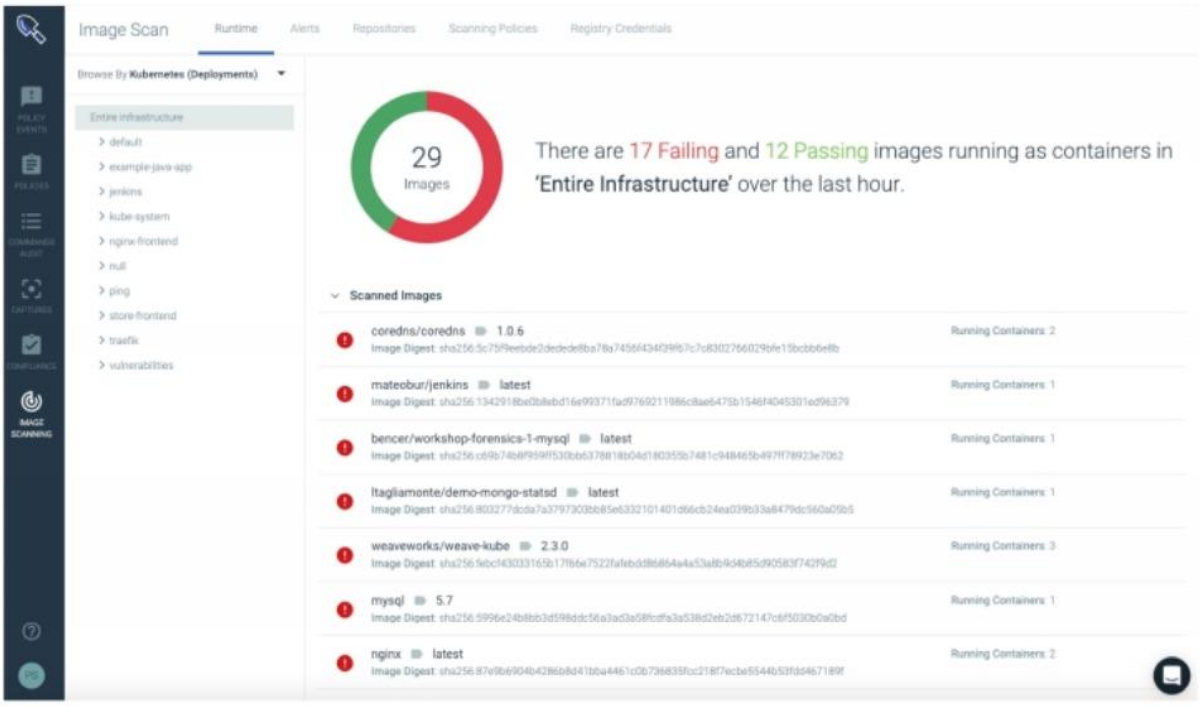
- Exploit poor security practices to perform data exfiltration and disclosure of sensitive information
- Gain unauthorized access to cloud applications.
- Misuse legitimate access to steal credentials of other cloud users.
- Generate illicit cryptocurrency by tapping into the target's processing power.
- Use stealthy crypto-mining malware to generate revenue
- Perform DoS attacks to prevent legitimate users from accessing cloud services.
- Reconfigure cloud services exploiting vulnerabilities in synchronization token systems.
- Perform lateral movement on data center networks and manipulate network traffic.

## Cloud Hacking Tools

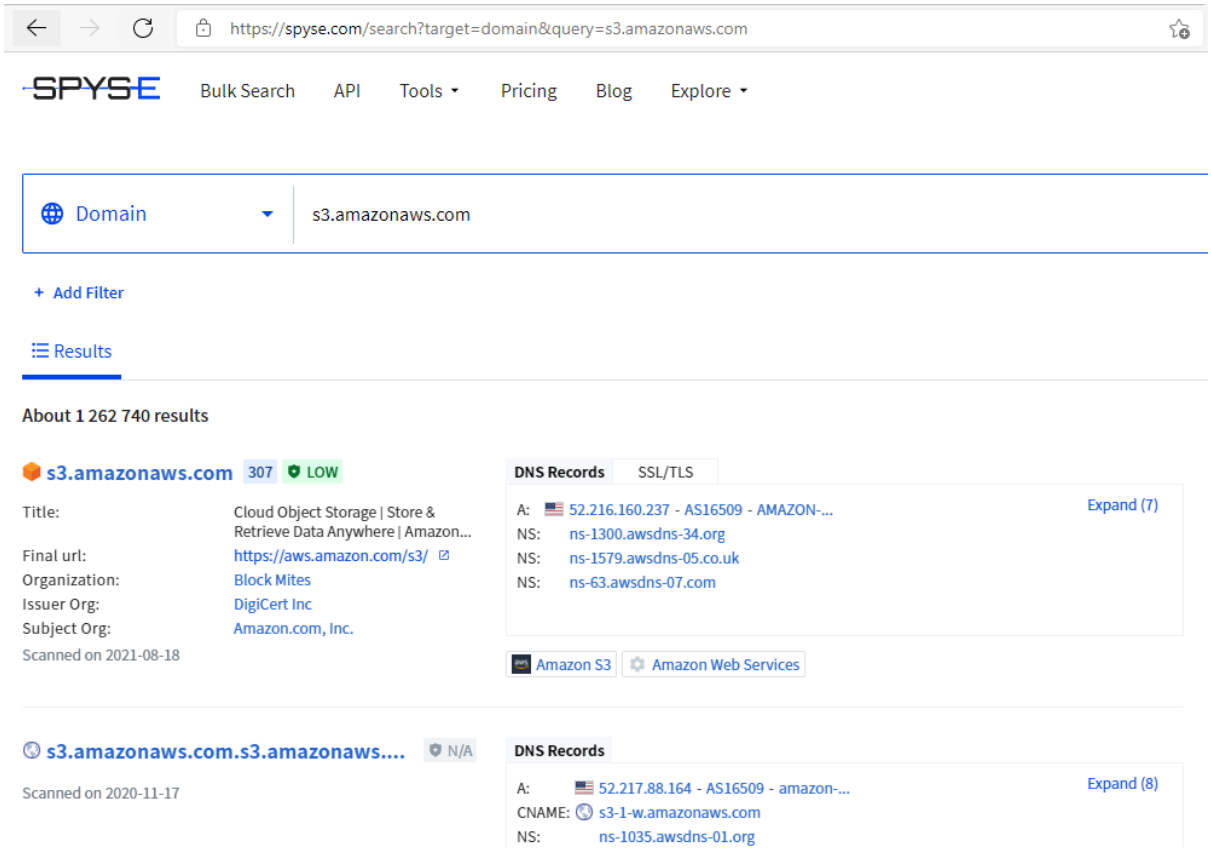
### Container Vulnerability Scanner using Trivy (https://github.com/aquasecurity/trivy)



## Kubernetes Vulnerability Scanner using Sysdig




## Enumerating s3 (aws) bucket using Spyse.com



## Several Advance Tools

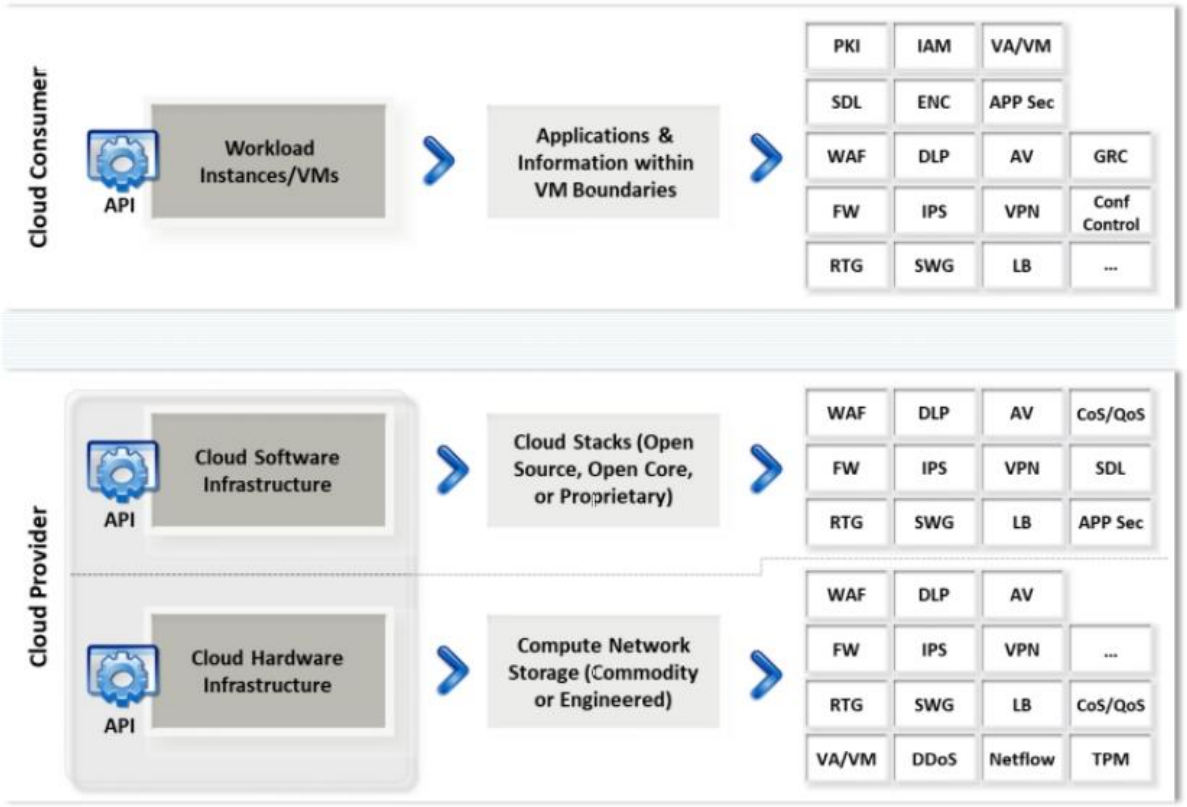
| | | |
|---|---|---|
| 1 | S3 Scanner | https://github.com/sa7mon/S3Scanner |
| 2 | S3 inspector | https://github.com/clario-tech/s3-inspector |
| 3 | Nimbostratus | https://andresriancho.github.io/nimbostratus/ |
| 4 | PACU | https://github.com/RhinoSecurityLabs/pacu |
| 5 | Dumpster Diver | https://github.com/securing/DumpsterDiver |
| 6 | Cloud Container Attack Tool | https://github.com/RhinoSecurityLabs/ccat |
| 7 | Cloud Goat AWS | https://rhinosecuritylabs.com/aws/cloudgoat-vulnerable-design-aws-environment/ |
| 8 | GCP Bucket Brute | https://github.com/RhinoSecurityLabs/GCPBucketBrute |
| 9 | Docker Scan | https://github.com/cr0hn/dockerscan |
| 10 | AWS PWN | https://github.com/dagrz/aws_pwn |

# Module Flow – Cloud Security

## Cloud security control layers

| | | |
|---|---|---|
| 1 | Application | SDLC, Binary Analysis, Scanners, Web App Firewalls, Transactional Sec |
| 2 | Information | DLP, CMF, Database Activity Monitoring, Encryption |
| 3 | Management | GRC, IAM, VA/VM, Patch Management, Configuration Management, Monitoring |
| 4 | Network | NIDS/NIPS, Firewalls, DPI, Anti-DDoS, QoS, DNSSEC, OAuth |
| 5 | Trusted Computing | Hardware and software RoT and API's |
| 6 | Computation and Storage | Host-based Firewalls, HIDS/HIPS, Integrity & File/Log Management, Encryption, Masking |
| 7 | Physical | Physical Plant Security, CCTV, Guards |

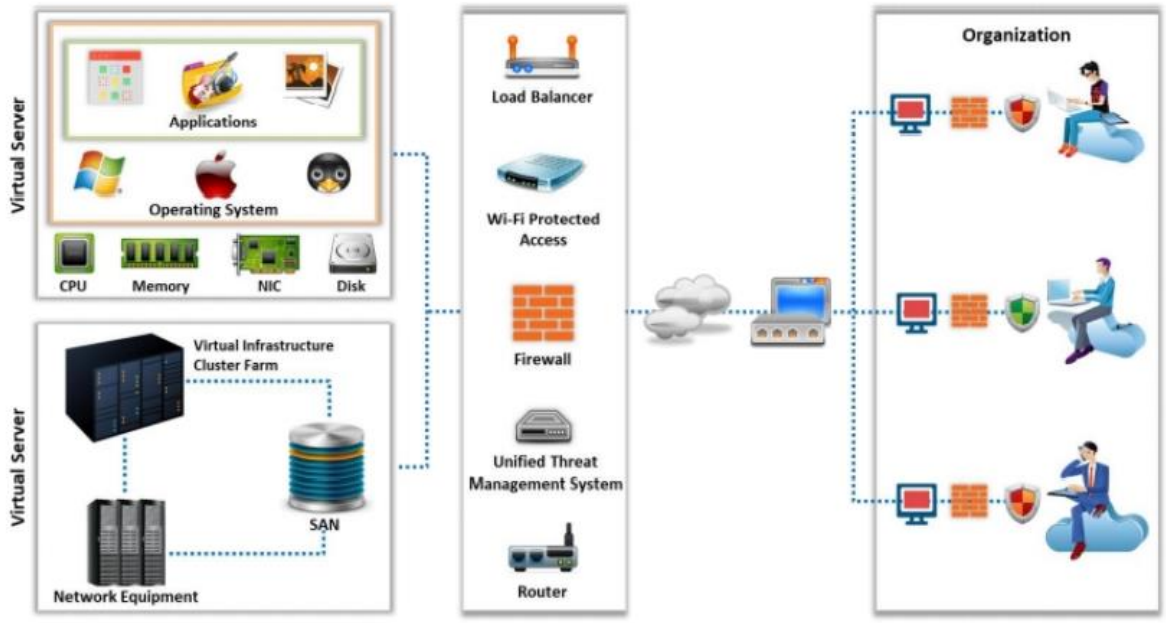
## Cloud Security Control Responsibility Provider and consumer



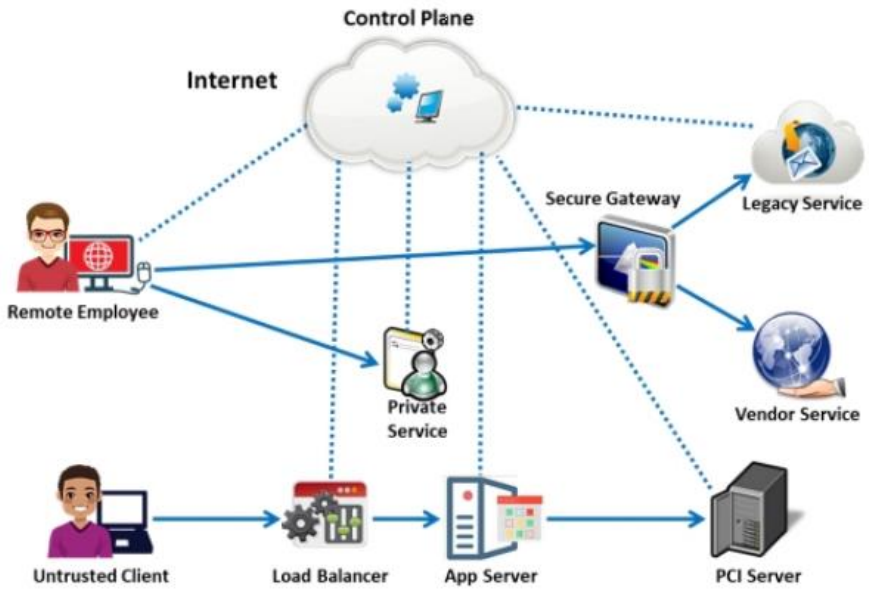
Following are some of the cloud security controls:

- **PKI**: Public key infrastructure
- **SDL**: Security development lifecycle
- **WAF**: Web application firewall
- **FW**: Firewall
- **RTG**: Real traffic grabber
- **IAM**: Identity and access management
- **ENC**: Encryption
- **DLP**: Data loss prevention
- **IPS**: Intrusion prevention system
- **SWG**: Secure web gateway
- **VA/VM**: Virtual application/Virtual machine
- **App Sec**: Application security
- **AV**: Anti-virus
- **VPN**: Virtual private network
- **LB**: Load balancer
- **GRC**: Governance, risk, and compliance
- **Config Control**: Configuration control
- **CoS/QoS**: Class of service/Quality of service
- **DDoS**: Distributed denial of service
- **TPM**: Trusted platform module
- **Netflow**: Network protocol by Cisco

## Placement security control in the cloud



## Implementation of Zero trust model




The Zero Trust model is a security implementation that assumes that every user trying to access the network is not a trusted entity by default and verifies every incoming connection before allowing access to the network.

It strictly follows the principle, "Trust no one and validate before providing a cloud service".

The idea behind implementing this model is to ensure a secured way of accessing resources to enforce strict access control and monitor the network traffic flow.

## Standard cloud security

### Open Web Application Security Project (OWASP)

The Open Web Application Security Project (OWASP) is an online community that produces freely-available articles, methodologies, documentation, tools, and technologies in the field of web application security.

- OWASP top 10 cloud security risks
- OWASP top 10 Serverless

### National Institute of Standards and Technology (NIST)

The National Institute of Standards and Technology (NIST) is a physical sciences laboratory and non-regulatory agency of the United States Department of Commerce. Its mission is to promote American innovation and industrial competitiveness. NIST's activities are organized into laboratory programs that include nanoscale science and technology, engineering, information technology, neutron research, material measurement, and physical measurement.

- NIST 800-190

## Cloud Security Alliance (https://cloudsecurityalliance.org/)

Cloud Security Alliance (CSA) is a not-for-profit organization with the mission to "promote the use of best practices for providing security assurance within cloud computing, and to provide education on the uses of cloud computing to help secure all other forms of computing."

## Security considerations

- Disaster recovery plan
- Service level agreement
- AAA
- Strong cryptography
- Multi-tenancy
- Quality of services
- Load balancing

## Kubernetes vulnerabilities and solutions

| Vulnerabilities | Solutions |
|---|---|
| 1. No Certificate Revocation | Ensure that nodes maintain the Certificate Revocation List (CRL)<br>Insist that administrators use OCSP stapling for revoking certificates |
| 2. Unauthenticated HTTPS Connections | Authenticate all HTTPS connections within the system<br>Ensure that all the components use CA maintained by the kube-apiserver<br>Implement two-way TLS for all the connections |
| 3. Exposed Bearer Tokens in Logs | Remove the bearer token from the system logs<br>Perform code reviews to ensure sensitive data is not logged and implement logging filters |
| 4. Exposure of Sensitive Data via Environment Variables | Avoid collecting sensitive data directly from environment variables<br>Use Kubernetes secrets in all components of the system |
| 5. Secrets at Rest not Encrypted by Default | Define and document configurations required for different levels of security |

| Vulnerabilities | Solutions |
|---|---|
| 6. Non-constant Time Password Comparison | Use a safe constant-time comparison function such as `crypto.subtle.ConstantTimeCompare`<br>Disapprove basic authentication mechanisms for secure options |
| 7. Hardcoded Credential Paths | Define a configuration method for credential paths, and avoid hardcoding credential paths<br>Allow cross-platform configuration through path generalization |
| 8. Log Rotation is not Atomic | Implement a copy-then-rename technique to ensure logs are not lost during log rotation<br>Avoid using log rotation, and implement persistent logs that add log data linearly |
| 9. No Back-off Process for Scheduling | Implement a back-off process for kube-scheduler to prevent tight-loops |
| 10. No Non-repudiation | Use secondary logging mechanisms for processes that require strict non-repudiation and auditing<br>All authentication events should be logged and retrievable from a central location |

## Serverless computing vulnerabilities and solutions

| Vulnerabilities | Solutions |
|---|---|
| **A1 - Injection** | Implement safe API, and employ parametrized interfaces or Object Relational Mapping Tools<br>Avoid special characters using a specific escape syntax in dynamic SQL queries |
| **A2 – Broken Authentication** | Employ identity and access control solutions<br>Implement strong authentication and access control on external-facing resources<br>Employ secure service authentication methods such as Federated Identity |
| **A3 – Sensitive Data Exposure** | Identify and classify sensitive data<br>Encrypt data both in transit and at rest<br>Implement HTTPS endpoints for APIs |
| **A4 – XML External Entities (XXE)** | Scan supply chain libraries for vulnerabilities<br>Test API calls for XXE vulnerabilities<br>Always disable Entity Resolution |
| **A5 – Broken Access Control** | Follow the least-privilege principle while granting permissions to functions |

| Vulnerabilities | Solutions |
|---|---|
| **A6 – Security Misconfiguration** | Use the cloud provider's built-in services such as AWS Trust Advisor, to identify public resources<br>Identify functions with unlinked triggers<br>Set the functions with a minimum timeout required |
| **A7 – Cross-Site Scripting (XSS)** | Encode all untrusted data before transmitting to the client<br>Use only well-known frameworks and headers |
| **A8 – Insecure Deserialization** | Ensure validation of serialized objects originating from untrusted data<br>Scan third-party libraries for deserialization vulnerabilities |
| **A9 – Using Components with Known Vulnerabilities** | Perform continuous monitoring of third-party libraries and dependencies<br>Deploy only signed packages and components from official sources |
| **A10 – Insufficient Logging and Monitoring** | Employ cloud service provider's monitoring tools such as Azure Monitor, or AWS CloudTrail to detect anomalous behavior |

## Best practice to securing the Cloud

1. Enforce **data protection**, **backup**, and **retention** mechanisms
2. Enforce **SLAs** for patching and vulnerability remediation
3. Vendors should regularly undergo **AICPA SAS 70 Type II audits**
4. Verify one's own cloud in **public domain blacklists**
5. Enforce **legal contracts** in employee behavior policy
6. Prohibit **user credentials sharing** among users, applications, and services
7. Implement strong **authentication**, **authorization** and **auditing** controls
8. Check for **data protection** at both the design stage and at runtime
9. Implement **strong key generation**, storage and management, and destruction practices
10. Monitor the **client's traffic** for any malicious activities
11. Prevent unauthorized server access using **security checkpoints**
12. Disclose applicable **logs** and **data** to customers

## American Institute of Certified Public Accountants (AICPA) Statement on Auditing Standards (SAS) 70 Type II

Statement on Auditing Standards (SAS) No. 70, Service Organizations, was a widely recognized auditing standard developed by the American Institute of Certified Public Accountants (AICPA). A service auditor's examination performed in accordance with SAS No. 70 (also commonly referred to as a "SAS 70 Audit") represents that a service organization has been through an in-depth examination of their control objectives and control activities, which often include controls over information technology and related processes.

13 Analyze **cloud provider security policies** and SLAs

14 Assess the security of **cloud APIs** and log customer **network traffic**

15 Ensure that the cloud undergoes regular **security checks and updates**

16 Ensure that physical security is a **24 x 7 x 365** affair

17 Enforce **security standards** in installation/configuration

18 Ensure that the memory, storage, and network access is **isolated**

19 Leverage strong **two-factor authentication** techniques where possible

20 Implement a baseline **security breach notification** process

21 Analyze **API dependency chain software** modules

22 Enforce stringent **registration and validation processes**

23 Perform vulnerability and configuration **risk assessments**

24 Disclose infrastructure information, **security patching**, and firewall details

25 Enforce stringent **cloud security compliance**, SCM (Software Configuration Management), and management practice transparency

26 Employ security devices such as IDS, IPS, and firewall to guard and stop **unauthorized access** to the data stored in the cloud

27 Enforce strict **supply chain** management and conduct a comprehensive supplier assessment

28 Enforce stringent **security policies and procedures** like access control policy, information security management policy, and contract policy

19 Ensure **infrastructure security** through proper management, and monitoring, availability, secure VM separation, and service assurance

30 Use **VPNs** to secure the client's data and ensure that data is **completely deleted** from the main servers along with its replicas when requested for data disposal

31 Ensure **Secure Sockets Layer** (SSL) is used for sensitive and confidential data transmission

32 Analyze the **security model** of the cloud provider interfaces

33 Understand the terms and conditions in the **SLA** like the **minimum level of uptime** and **penalties** in case of failure to adhere to the agreed level

34 Enforce basic information security practices such as strong **password policy**, **physical security**, device security, **encryption**, data security, and network security

## NIST Recommendation for cloud security

1. **Assess the risk** posed to the client's data, software and infrastructure
2. Select an appropriate **deployment model** according to the needs
3. Ensure **audit procedures** are in place for data protection and software isolation
4. **Renew SLAs** in case of **security gaps** found between the organization's security requirements and the cloud provider's standards
5. Establish appropriate **incident detection** and **reporting mechanisms**
6. Analyze what are the **security objectives** of the organization
7. Enquire about **who is responsible** for data privacy and security issues in the cloud

## Best Practices for Container Security

1. Regularly **monitor the CVEs** of the container runtime and remediate, if any vulnerabilities are detected
2. Employ **app-aware tools** to monitor container network interfaces, network traffic, and network anomalies
3. Configure applications to **run as normal users** to prevent privilege escalation
4. Configure the **host's root file system in read-only mode** to restrict the write access
5. Employ **application security scanning tools** to protect containers from malicious software
6. Perform regular **scanning of the images** in the repository to identify vulnerabilities or misconfigurations
7. Deploy **application firewalls** for enhancing container security and prevent threats entering the environment
8. Use a **separate database for each application** for greater visibility of individual applications
9. Regularly **update the host operating system** and the kernel to the latest security patches
10. **Configure orchestrators** to deploy sets of hosts separately based on their sensitivity level
11. **Automate the compliance** to the container runtime configuration standards
12. **Maintain a set of trusted registries and images** and ensure only images from this set are permitted to run in the container environment

## Best Practices for docker Security

1. Avoid exposing the Docker daemon socket
2. Always use trusted Docker images only
3. Regularly patch the host OS and Docker with the latest security updates
4. Limit the capabilities by allowing access only to the features required by the container
5. Always run the Docker images with **--security-opt=no-new-privileges** to prevent privilege escalation attacks
6. Disable the inter-container communication feature for running Docker demon using **--icc=false**
7. Use Linux security modules such as **seccomp**, **AppArmor**, and **SELinux** to gain fine-grained control over the processes
8. Enable read-only mode on file systems and volumes by setting **--read-only** flag
9. Set the Docker daemon log level to '**info**' and avoid running the Docker daemon using the '**debug**' log level
10. Configure the container application to run as an unprivileged user to prevent privilege escalation attacks
11. Check that Docker images from the remote registry are digitally signed using the Docker content trust
12. Secure the API endpoints with HTTPS when exposing RESTful API
13. Always store sensitive data in Docker volumes for enhanced data security
14. Use tools such as **InSpec** and **DevSec** to detect Docker vulnerabilities

## Best Practices for Kubernetes Security

1. Ensure proper validation of **file contents** and their path at every stage of processing
2. Implement the **configuration method** for the credential paths and do not depend on the hardcoded paths
3. Raise errors explicitly after each step of a **compound operation**
4. Use the **copy-then-rename method** for logs rotation to ensure that the logs are not lost when restarting the kubelet
5. Use the well-tested JSON library and type structures for constructing JSON objects
6. Never use compound shell commands without proper validations
7. Check the returned error value of **os.Readlink /proc/<pid>/exe** explicitly to determine if the PID is a kernel process
8. Use common parsing functions such as ParsePort across the codebase to **increase code readability**
9. Limit the size of manifest files to prevent **Out-Of-Memory errors** in kubelet
10. Use **kube-apiserver instances** that maintain CRLs to check the presented certificate
11. Use KMS to **enable secret data encryption** and avoid using AES-CBC or GCM for encryption
12. Avoid using **legacy Secure Shell** (SSH) tunnels as it does not perform proper validation of the server IP address
13. Use **OSCP stapling** to check the revocation status of the certificates
14. **Use TLS** in development and production configurations to reduce the vulnerabilities due to misconfiguration

## Best Practices for Serverless Security

1. **Minimize serverless permissions** in the development phase to reduce the attack surface area
2. Regularly monitor function layers to identify malicious code injection attempts
3. Use **third-party security tools** as they provide additional layers of visibility and control
4. Regularly patch and update **function dependencies** and applications
5. Use tools such as **Snyk** to scan serverless applications for known vulnerabilities
6. Properly **sanitize event input** to prevent code injection attacks
7. Use security libraries that disable access to resources and implement **runtime least-privileges**
8. Deploy functions in minimal granularity to minimize the level of detail and to **prevent implicit global roles**
9. **Employ data validation** technique on schemas and objects instead of data serialization and deserialization
10. Leverage **API gateway capabilities** to perform input data filtering, traffic throttling, and rate limiting
11. Audit and monitor functions by enforcing verbose and safe logging of function events
12. Use secure coding practices and perform **code review sessions** to patch the vulnerable application code

## Organization/provider cloud security compliance checklist

| Management | Organization | Provider |
|---|---|---|
| Is everyone aware of his or her cloud security responsibilities? | | |
| Is there a mechanism for assessing the security of a cloud service? | | |
| Does the business governance mitigate the security risks that can result from cloud-based "shadow IT"? | | |
| Does the organization know within which jurisdictions its data can reside? | | |
| Is there a mechanism for managing cloud-related risks? | | |
| Does the organization understand the data architecture needed to operate with appropriate security at all levels? | | |
| Can the organization be confident of end-to-end service continuity across several cloud service providers? | | |
| Does the provider comply with all relevant industry standards (e.g. the UK's Data Protection Act)? | | |
| Does the compliance function understand the specific regulatory issues pertaining to the organization's adoption of cloud services? | | |

## Cloud Security Tools

- CipherCloud (https://www.ciphercloud.com)
- Netskope Security Cloud (https://www.netskope.com)
- Prisma Cloud (https://www.paloaltonetworks.com)
- ForgeRock Identity Cloud (https://www.forgerock.com)
- Deep Security (https://www.trendmicro.com)

## Tools for securing containers

- Sysdig Falco (https://sysdig.com)
- Anchore (https://anchore.com)
- NeuVector (https://neuvector.com)
- Lacework (https://www.lacework.com)
- Tenable.io Container Security (https://www.tenable.com)

## Tools for securing Serverless Computing

- Synk (https://snyk.io)
- Twistlock Serverless Security (https://www.twistlock.com)

- Aqua (https://www.aquasec.com)
- Prisma Cloud (https://www.paloaltonetworks.com)
- Dashbird (https://dashbird.io)