SE 464
Week 9

Secure Microservices,
Publisher-Subscriber Architecture

Secure Microservices

Microservices Security Challenges

- Involves numerous small services, each handling specific business tasks like billing
- Security concerns: access control, secure communication, and protection against data theft or unauthorized access
- Goal: controlling access and securing communication routes

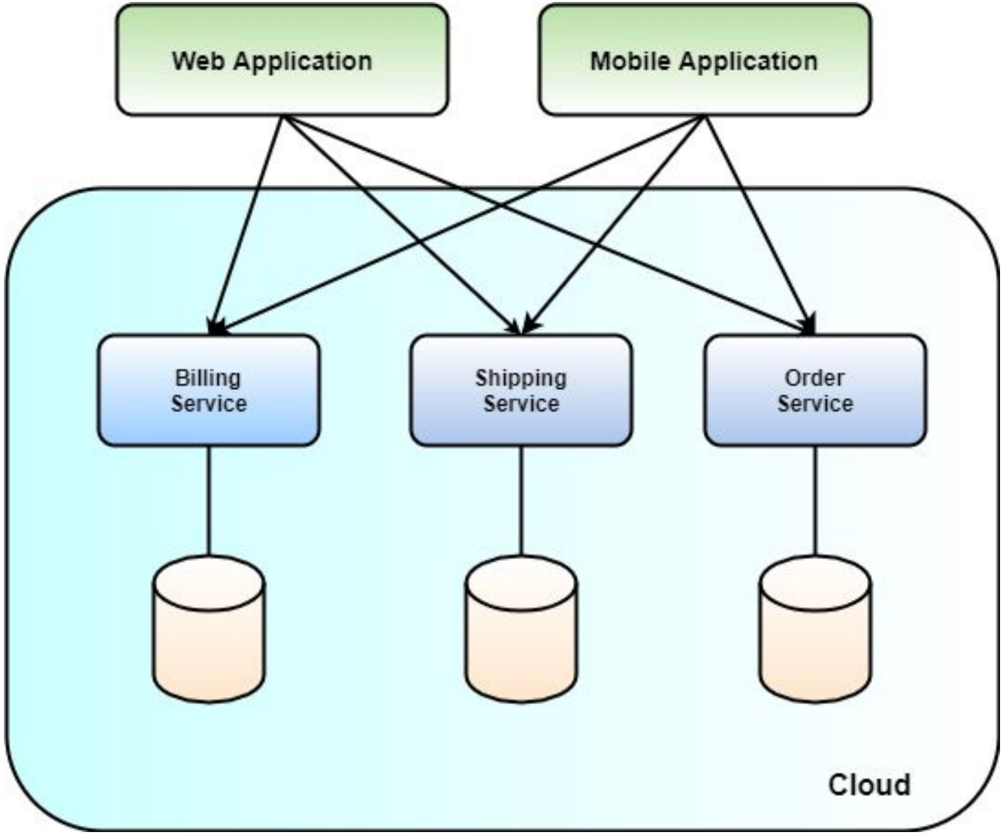
Objective

- Protect internal services from public access
- Rationale for Protection:
- Minimizing attack surface
- Preventing direct public access to services

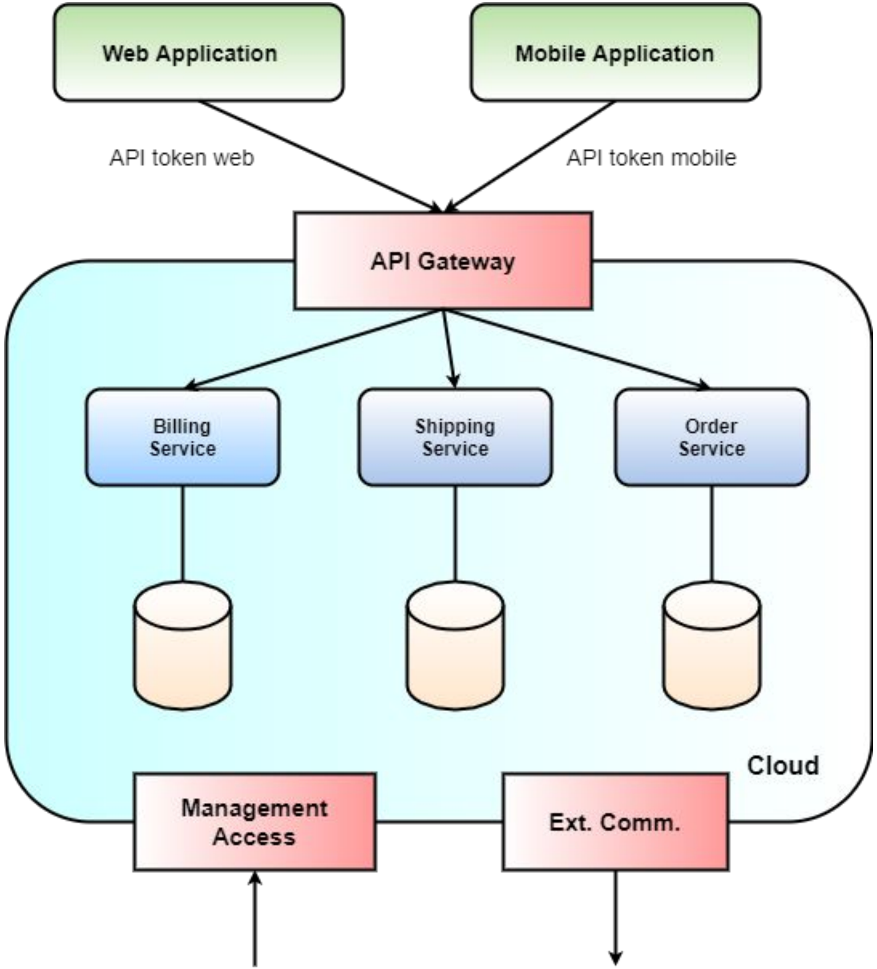
Consequences of Inadequate Protection

- Increased security overhead for each service
- Higher operational costs
- Easier target identification for attackers
- Potential for escalated attacks upon service compromise

Bad Example - Cluster w/o Entry Point



Good Example - Cluster with Entry Point



Role of the API Gateway

- API Gateway as Entry Point:
  - Defines exposed routes to client applications
  - Differentiates between public and internal APIs
- Reducing Attack Surface:
  - Expose only necessary routes and endpoints
  - Minimizes potential targets for attackers

Implementing Security through API Gateway

- Global Authentication Techniques:
  - Enforces API access tokens for authentication
  - Verifies pre-shared secrets for each client application
- Differentiation of Application Access:
  - Unique API access token for each application
  - Helps in limiting API access based on client application
- Security Strategies:
  - Implements rate-limiting to prevent denial-of-service attacks
  - Mitigates overload of microservices by external requests

Further Security Measures

- Management Security:
  - Enforces strict access rules for managing infrastructure
  - Integrates with Active Directory or similar services for control plane protection
- Protecting Outbound Communication:
  - Secures communication with external dependencies

Significance of Certificate Verification in Security

- Verification to Prevent Man-in-the-Middle Attacks:
  - Ensures client communicates with the intended server
  - Distinguishes between malicious and legitimate servers

Securing Communications

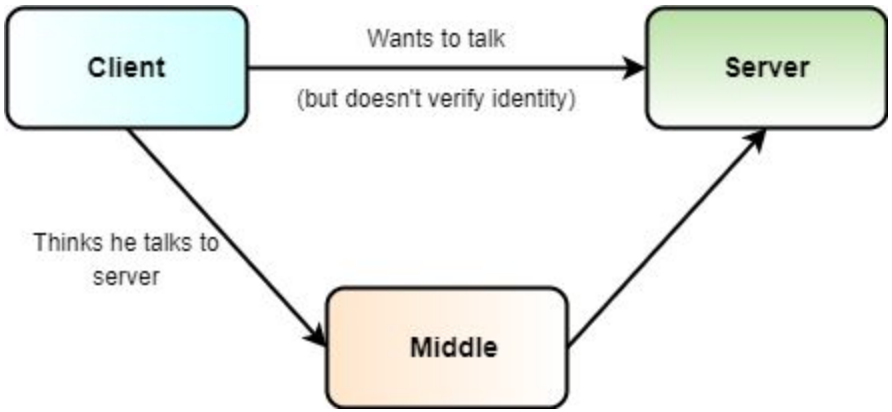
- Use of HTTPS/TLS (Transport Layer Security):
  - Encrypts and authenticates communication channels
  - Ensures confidentiality and authenticity of data in transit
- TLS Protocol Mechanics:
  - Involves a handshake between communicating parties
  - Server presents a certificate containing a public key

Significance of Certificate Verification in Security

- Verification to Prevent Man-in-the-Middle Attacks:
    - Ensures client communicates with the intended server
    - Distinguishes between malicious and legitimate servers

Significance of Certificate Verification in Security

- Man in the middle attack:



TLS Certificate and Verification Process

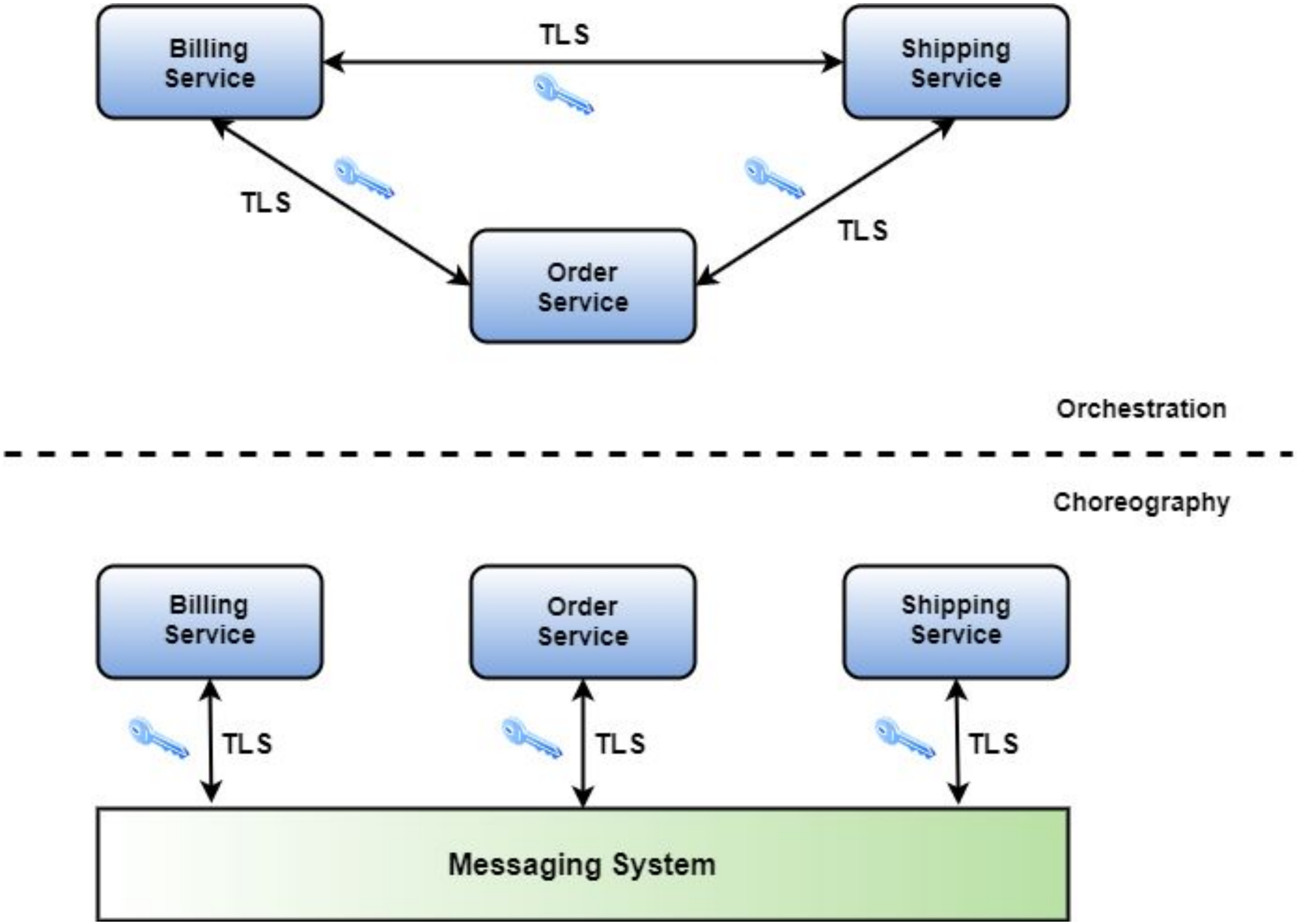
- Certificate Components:
  - Public key enclosed in a signed envelope
  - Signature by issuing authority using a private key
- Importance of Private Key Security:
  - Leakage of private keys compromises certificate authenticity
- Client's Certificate Verification Strategies:
  - Accept all certificates
  - Accept self-signed certificates
  - Accept certificates signed by trusted authorities
  - Accept only a specific certificate (certificate pinning)

TLS Certificate Verification Options

- From previous slide:
- Option 1 and 2:
  - No security checks or trust verification
  - Self-signed certificates without external validation
- Option 3 and 4:
  - Verification against trusted authorities
  - Acceptance of specific, trusted certificates only

Mutual TLS and Client Authentication

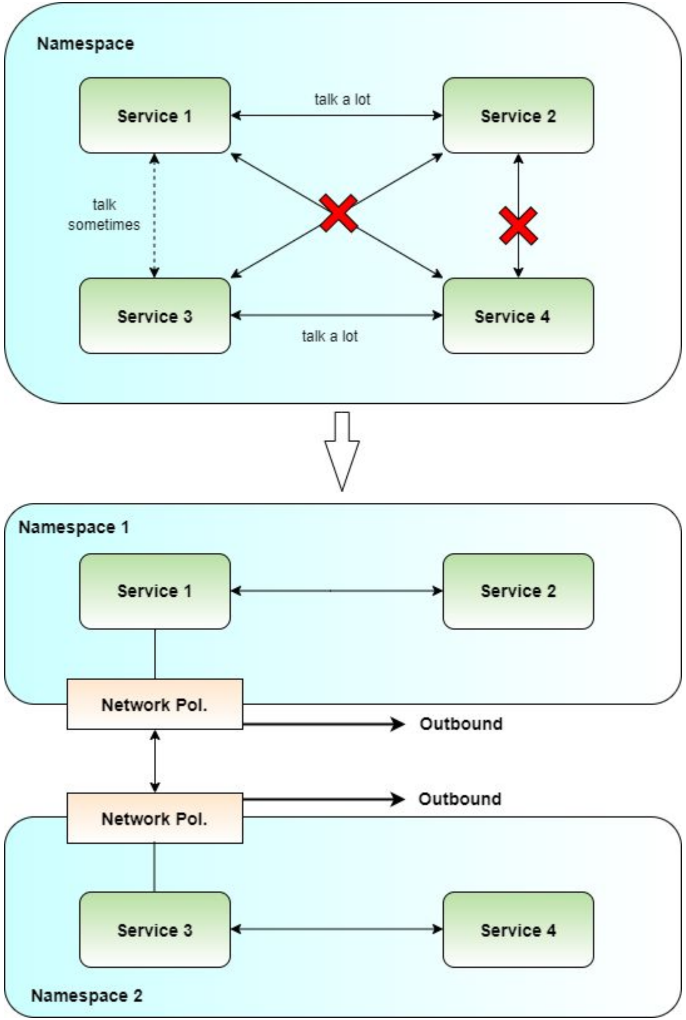
- Standard TLS:
  - Only the client verifies the server certificate
- Mutual TLS (mTLS):
  - Both client and server present and verify certificates
  - Utilizes similar verification strategies as standard TLS



Billing Service
TLS
Shipping Service
TLS
TLS
Order Service
Orchestration
Choreography
Billing Service
Order Service
Shipping Service
TLS
TLS
TLS
Messaging System

Internal Security within Clusters

- Want to avoid different services talking directly to one another
- Need to limit the damage that can be done in scenario when intruder breaks into your cluster



- Communication between namespaces

Implementing Network Segmentation in Kubernetes

- Use of Namespaces:
  - Groups cluster into smaller, manageable units
  - Each namespace acts like a distinct segment
- Network Policies in Kubernetes:
  - Regulate traffic between namespaces
  - Define rules for inbound and outbound traffic

Importance of Outbound Communication Policies

- Outbound Traffic Control:
  - Prevents misuse of services for attacks like SSRF
  - Protects against installation of malicious software in Pods
- Potential Risks of Neglecting Outbound Policies:
  - Exploitation for server-side request forgery
  - Unauthorized software installation in compromised Pods

Monitoring Clusters

- Behavior Monitoring Frameworks:
  - Examples include Falco
  - Define expected behavior via configuration files
- Security Alerts:
  - Frameworks notify operators on deviation from normal behavior
  - Enhances detection of suspicious activities within the cluster

Resource Management for Security

- Resource Quotas in Kubernetes:
  - Sets limits on resources like memory, CPU per namespace
- Preventing Denial-of-Service Attacks:
  - Limits resource consumption to avoid service disruption
  - Essential for maintaining service availability

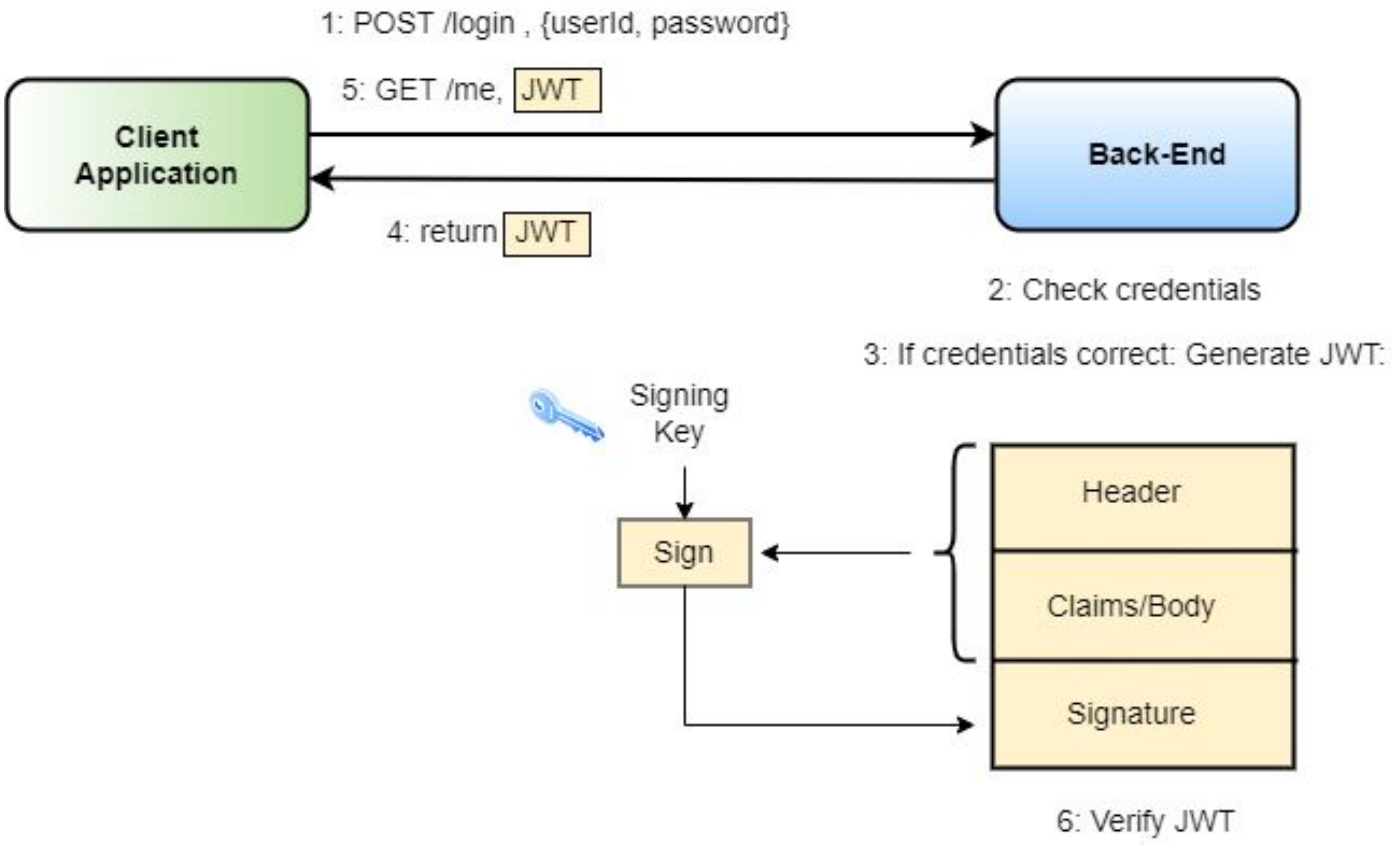
Securing Individual Microservices

- Previous slides discussed securing entire clusters
- Focus on Protecting Microservices:
  - Each service requires multiple layers of security

Identity Verification in Microservices

- Using JSON Web Tokens (JWTs):
  - Consists of a header, body, and signature
  - Header specifies signing algorithm
  - Body contains user claims and token validity
  - Signature ensures token integrity and authenticity

Creating JWTs:

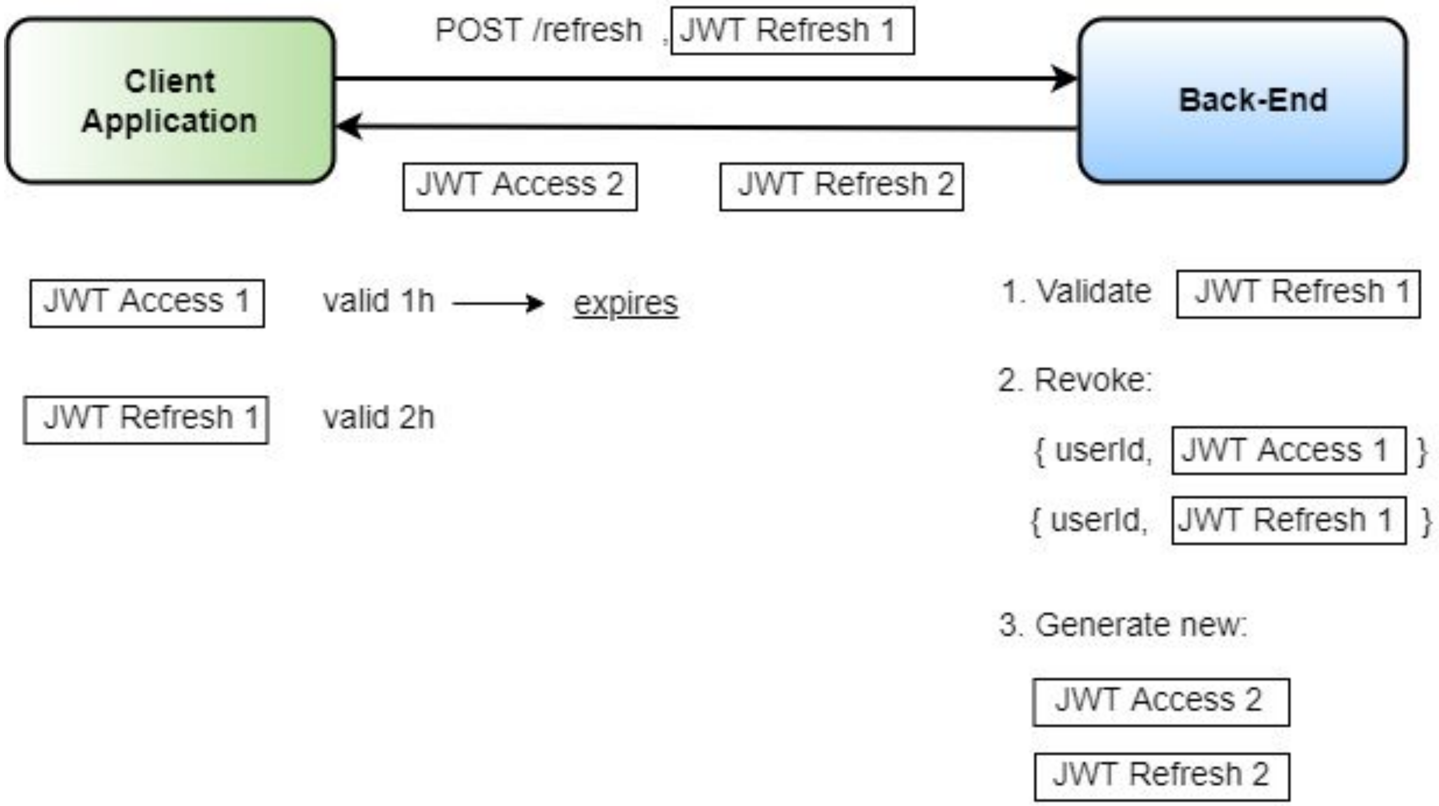


JWT Lifecycle Management

- Token Validity and Renewal:
  - Balancing token lifespan for security and usability
  - Use of refresh tokens for longer validity
- Token Issuance Process:
  - User credentials verified by authorization service
  - Successful validation leads to JWT issuance

Handling Access and Refresh Tokens

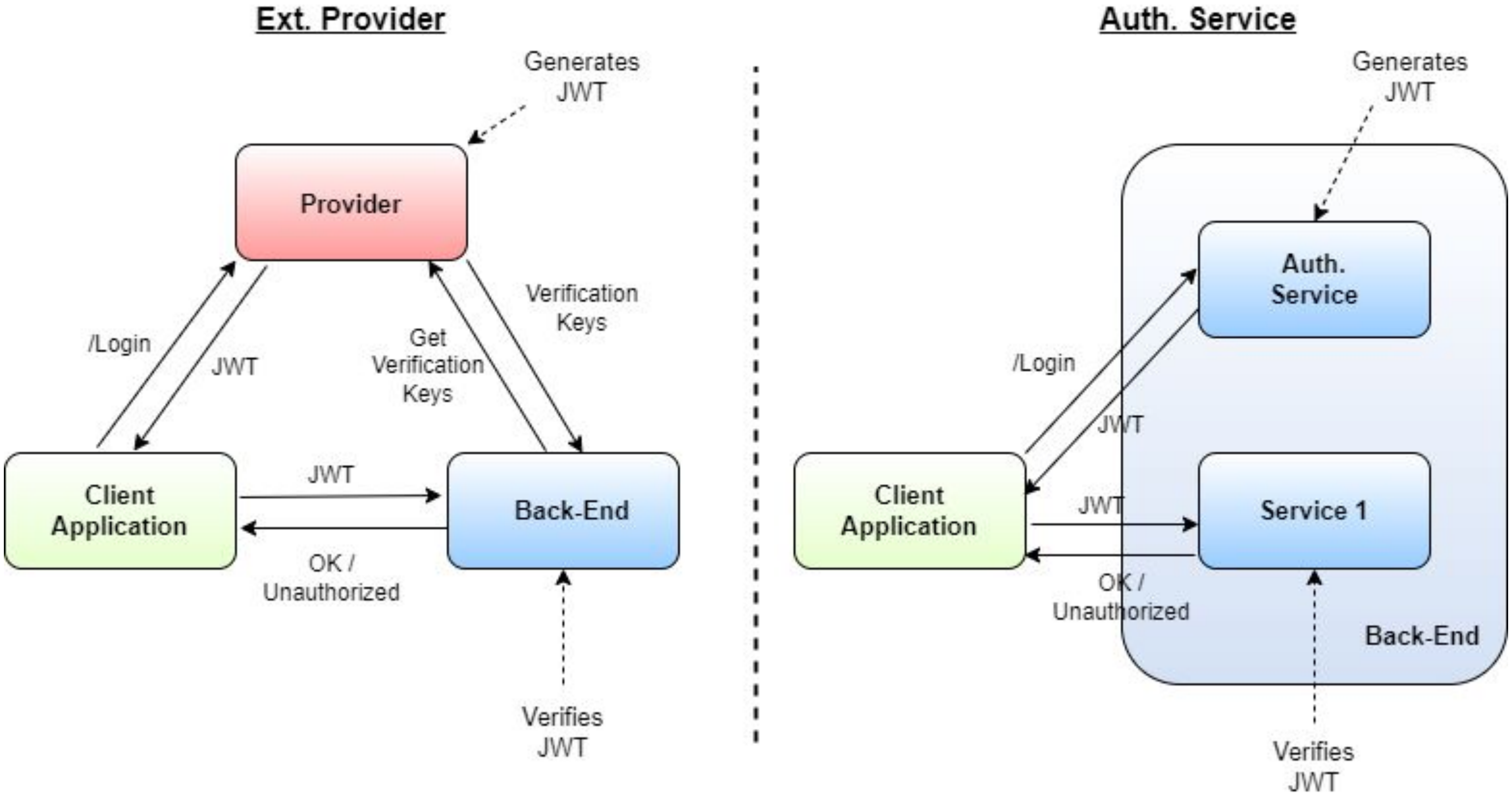
- Token Management Strategy:
  - Issuance of both access and refresh tokens
  - Revoking and renewing tokens upon expiration
- Minimizing Risks of Token Theft:
  - Automatic invalidation of all user tokens if one is compromised


POST /refresh , JWT Refresh 1
Client Application
Back-End
JWT Access 2
JWT Refresh 2
JWT Access 1
valid 1h
expires
JWT Refresh 1
valid 2h
1. Validate JWT Refresh 1
2. Revoke:
{ userId, JWT Access 1 }
{ userId, JWT Refresh 1 }
3. Generate new:
JWT Access 2
JWT Refresh 2

Options for Token Management in Microservices

- Option A:
  - Using external provider for token generation
  - Token validation at API gateway or individually by services
- Option B:
  - Each service manages its own token generation and validation
- Option C:
  - Centralized token generation; individual services validate tokens
  - Asymmetric cryptography with secret key only known to the authorization service

Internal vs External Authentication



Port Management and Security

- Minimizing Open Ports:
  - Only essential ports should be open
  - Avoid unnecessary management interfaces
- Risks of Additional Ports:
  - Each open port increases attack surface
  - Management interfaces are high-risk targets

Implementing Access-Control Lists

- Selective Visitor Acceptance:
  - Use access-control lists for critical services
- Enforcement Mechanism:
  - API gateway enriches HTTP headers with source IP
  - Services check IP against Allow list for access

Information Leakage Prevention

- Sanitizing Errors and Exceptions:
  - Prevent leakage of internal back-end details
- Mitigation of Attack Surface Expansion:
  - Avoid detailed error messages that reveal system information

Middleware

- Use of Service Chassis:
  - Middleware for cross-cutting security concerns
  - Recommended order: Exception, Authentication, Authorization, Access Control
- Optional Rate-Limiting Middleware:
  - Apply based on source IP or user identity

Other Security Strategies

- Docker Container Hardening:
  - Restrict ports for communication
  - Avoid running containers as root user
  - Create a new user in Dockerfile, drop unnecessary rights
- Input Data Validation:
  - Essential for preventing unauthorized data manipulation
  - Overlooked but critical for service security

Input Data Validation Risks

- SQL Injection Risk:
  - Example: Billing service endpoint querying credit card information
  - Risk from string concatenation in SQL queries
- Remote Code Execution (RCE) Vulnerability:
  - Example: Unsanitized input strings in .NET process invocation
  - Potential for attacker-controlled process execution or reverse shell creation
- Cross-Site Scripting (XSS) and Other Risks:
  - Data passed between services without sanitization
  - Risks in front-end usage leading to XSS vulnerabilities

Solutions for Validating Input Data

- Validation Techniques:
  - Validate integer ranges
  - Validate string usage contextually
  - Ensure data transfer object deserialization is secure
- Context-Specific String Validation:
  - Database queries: Prevent SQL injection via OR mappers or prepared statements
  - Webpages: Escape strings to mitigate XSS
  - URLs: Validate against allow list, sanitize for SSRF prevention
  - Email services: Verify email format and recipient validity

Summary

- Initial Protection Measures:
  - Secure cloud environment with a virtual private cloud
  - Restrict access through a single, well-guarded entry point
- Communication Security:
  - Implement TLS for protected communication channels
  - Prevent unauthorized data reading or modification within the cluster
- Cluster Segmentation:
  - Divide the cluster into smaller segments for controlled communication
  - Apply strict pathways for inter-service communication

Summary (cont.)

- Service-Level Protection:
  - Use JWT-based authentication and access-control lists
  - Middleware implementation for reusability and uniformity
- Hardening and Validation Techniques:
  - Docker container hardening
  - Essential input data validation and sanitizing

Security Case Study:
Cybersecurity Incident Simulation @ Uber

Importance of Cybersecurity Simulations

- Preparedness for Incidents:
  - Essential for coordinated response in real incidents
  - Tests ability of key people and functions to act effectively
- Benefits of Simulations:
  - Enhances readiness for cybersecurity incidents
  - Provides practical experience in incident management

Approach

- Diverse Simulation Approaches:
  - Various methods offer different benefits and limitations
  - Balance between realism and frequency of simulations
- Simulation Planning and Execution:
  - Sophisticated simulations require extensive planning
  - Frequent, less complex simulations for regular practice
- Combining Simulation Types:
  - Array of options for comprehensive incident response readiness
  - Tailors simulation strategy to organizational needs

Three Main Simulation Methods

- Tabletop Exercises (TTX)
- Red Team Operations
- Atomic Simulations

Tabletop Exercises (TTX)

- Objective:
  - Simulate security incidents for process and decision-making improvement
- Key Goals:
  - Enhance large-scale incident response capabilities
  - Increase cybersecurity awareness among executive leadership
  - Evaluate leadership team’s decision-making in crisis
- TTX Format:
  - Shift from traditional scripted format to realistic role-playing
  - Involvement of a virtual Security Operations Center (vSOC)
  - Use of “injects” to guide and challenge the Cyber Incident Response Team (CIRT)

Red Team Operations at Uber

- Approach:
  - High planning overhead, mimics real-world threat actor activity
  - Focus on intrusion to action or network eviction stages
- Key Activities:
  - Annual capture the flag event for cross-company team collaboration
  - Unannounced operations treated as real incidents
- Special Event:
  - Annual Red vs. Blue event, involving key stakeholders for intensive simulation

Atomic Simulations

- Purpose:
  - Test smaller-scale, realistic incident scenarios
  - Focus on detection, standard operating procedures (SOPs), and threat intelligence
- Advantages:
  - Low overhead, repeatable
  - Quick retesting to gauge effectiveness of improvements

Atomic Simulations (cont.)

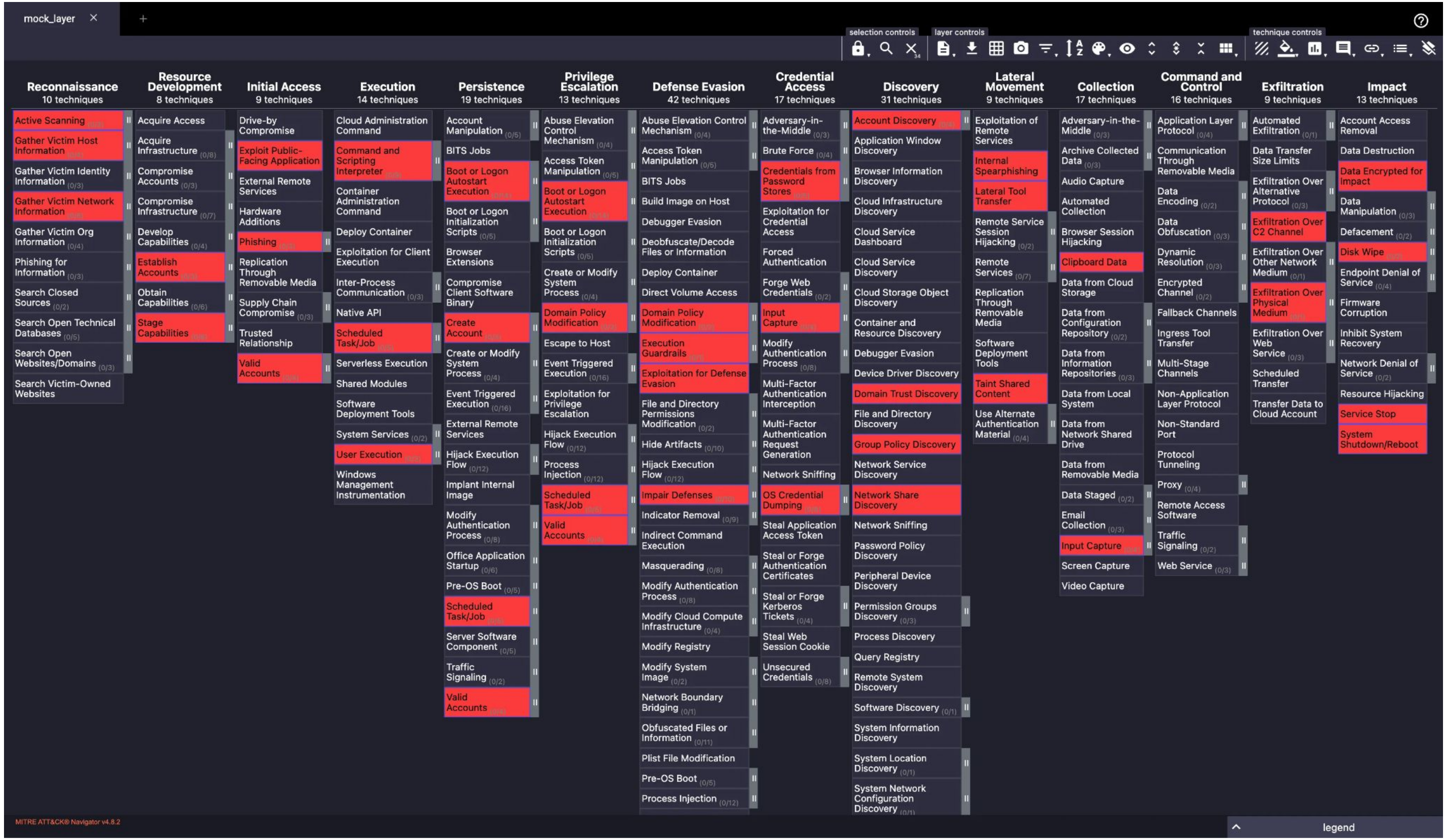
- Process:
  - Execute chain of tactics, techniques, and procedures (TTPs)
  - Simulate threat actor paths including RAT deployment, network reconnaissance, and data exfiltration
- After-Action Review:
  - Discuss gaps and improvements post-simulation

Summary

	Red Team & Red/Blue Exercises	TTX	Atomic Incident Sim
Frequency	Quarterly with a CTF once a year	4 times a year	Twice a month
Exercises Process	Yes	Yes	Yes
Technical Investigation	Yes	No hands-on investigation	Yes
Exercises Communications	Yes	Yes	Yes
Executive Engagement	Yes	Yes	Unlikely

Comprehensive Approach

- Three-Pronged Approach:
  - Offers a broad way to test security posture
- Coverage Tracking:
  - Across different environments (corp, prod, cloud, etc.)
  - Utilization of MITRE ATT&CK® Navigator:
  - Maps simulations to Tactics, Techniques, and Procedures (TTPs)
  - Tracks the range of simulated scenarios

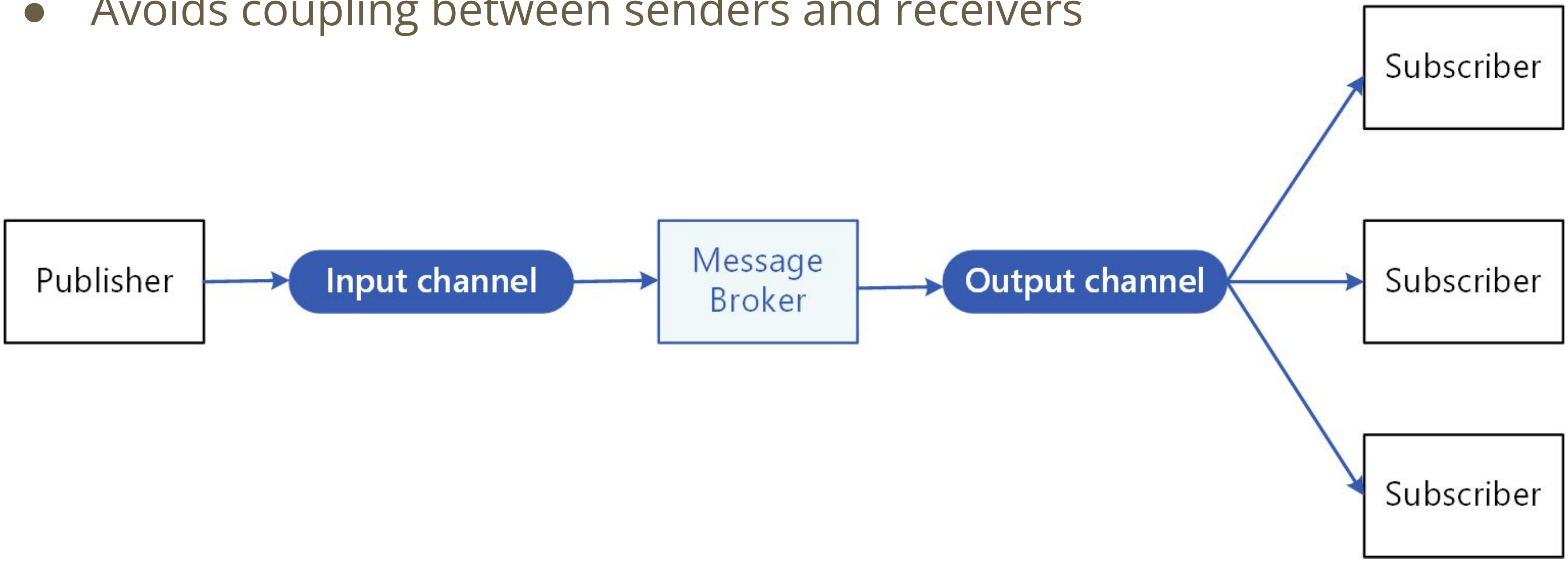


Fictional example ATT&CK Matrix showing TTPs that have been simulated

Publisher-Subscriber Architecture

Purpose

- Enables applications to announce events asynchronously
- Avoids coupling between senders and receivers



Challenges

- Need in Distributed Systems:
  - Components provide information to others as events occur
- Advantages of Async:
  - Decouples senders from consumers
  - Prevents sender blockage waiting for responses
- Scaling Challenges:
  - Inefficiency with dedicated message queues for each consumer
  - Addressing varying consumer interests in event information
- Primary Question:
  - How to announce events to all interested consumers without knowing their identities?

Asynchronous Messaging Subsystem

- Key Components:
  - Input messaging channel for sender (publisher)
  - Output messaging channel per consumer (subscriber)
  - Intermediary for message distribution (message broker/event bus)
- Message and Event Definitions:
  - Message: Packet of data
  - Event: Message indicating a change or action

Benefits

- Subsystem Decoupling:
  - Independent management of subsystems
  - Continuous message management despite offline receivers
- Enhanced Scalability and Sender Responsiveness:
  - Sender quickly dispatches messages, returns to core tasks
  - Messaging infrastructure ensures delivery to subscribers

Benefits

- Reliability Improvements:
  - Smooth operation under load
  - Effective handling of intermittent failures
- Deferred or Scheduled Processing:
  - Subscribers process messages during off-peak hours
  - Message routing based on specific schedules

Benefits

- Integration Across Diverse Systems:
  - Facilitates communication between different platforms, languages, protocols
  - Integrates on-premises and cloud applications
- Workflow Facilitation and Testing:
  - Supports asynchronous workflows across an enterprise
  - Enables monitoring and inspection of channels for integration testing
- Separation of Concerns:
  - Applications focus on core capabilities
  - Messaging infrastructure handles reliable routing to multiple consumers

Considerations for Implementation

- Choice of Technology:
  - Utilize existing messaging products like Azure Service Bus, Event Hubs, Redis, RabbitMQ, Apache Kafka
- Subscription Handling:
  - Messaging infrastructure to allow easy subscription and unsubscription
- Security Measures:
  - Implement security policies to restrict unauthorized channel access

Message Distribution

- Handling Message Subsets:
  - Use of topics for dedicated output channels
  - Content filtering based on message content
  - Wildcard subscribers for multiple topics
- Bi-Directional Communication:
  - Use Request/Reply Pattern for acknowledgments or status communication

Message Ordering

- Message Ordering Challenges:
  - Design for idempotent message processing
  - No guarantee of message reception order
- Managing Message Priority:
  - Implement Priority Queue pattern for ordered message processing

Handling Poison Messages

- Dealing with Poison Messages:
  - Prevent return to queue, store separately for analysis
  - Utilize dead-letter queue functionality in message brokers
- Duplicate Message Handling:
  - Implement duplicate detection and removal (de-duping)
  - Ensure idempotent processing if infrastructure doesn't de-duplicate

Message Expiration and Scheduling

- Message Expiration:
  - Implement limited lifetime for messages
  - Include expiration time in message data
- Message Scheduling:
  - Enable embargo on messages until specific date and time
  - Restrict receiver access until specified processing time

When to Use

- Broad Information Broadcasting:
  - Suitable for applications needing to reach numerous consumers
- Cross-Platform Communication:
  - Ideal for interacting with diverse applications or services
  - Supports varied platforms, languages, and protocols
- Non-Real-Time Communication:
  - Effective when immediate consumer responses are not required

Consistency and Availability

- Support for Eventual Consistency:
  - Applicable for systems with eventual data consistency models
- Varied Consumer Availability:
  - Useful for communicating with consumers having different uptime schedules

Robotics Operating System (ROS)

Intro to ROS

- Overview:
  - Flexible framework for writing robot software
  - Collection of tools, libraries, and conventions for simplifying complex robotic behaviors
- Development and Community:
  - Open-source project
  - Large community contributing to its extensive library
- Core Philosophy:
  - Peer-to-peer structure
  - multi-language integration
  - Thin tools/libraries layering

Use Cases

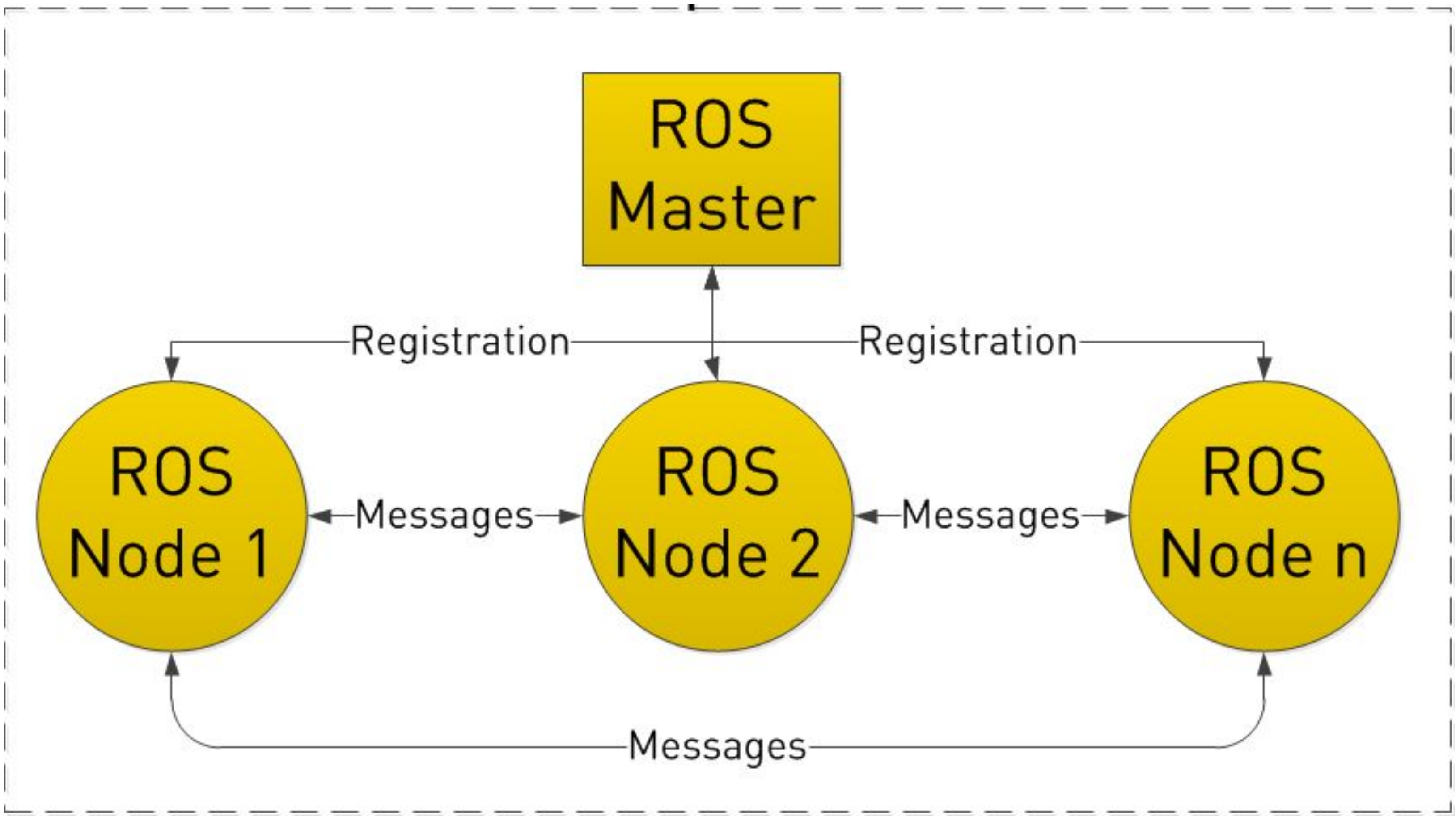
- Industrial Automation:
  - Complex assembly and automation tasks
- Research and Education:
  - Platform for robotics research and learning
- Space Exploration:
  - Remote operation and data collection in harsh environments
- Healthcare Robotics:
  - Assisted surgery and rehabilitation tools

Advantages

- Modularity:
  - Reusable components for various tasks
- Scalability:
  - Suits small and large-scale robotic systems
- Flexibility:
  - Supports multiple hardware and software configurations

ROS Architecture

- Three Key Elements:
  - Nodes: Processes performing computation
  - Messages: ROS data structure for communication
  - Topics: Named buses over which nodes exchange messages
- Additional Components:
  - Services: Synchronous remote procedure call
  - Master: Name service for ROS (helps nodes find each other)
  - Rosout: ROS equivalent of stdout/stderr


ROS Master
Registration
Registration
ROS Node 1
Messages
ROS Node 2
Messages
ROS Node n
Messages

ROS Master

- Provides naming and registration services to the rest of the nodes
- Tracks publishers and subscribers
- Essential for node communication setup
- Centralized point for network configuration

ROS Nodes

- Fundamental building block of ROS application
- Single-purpose executable
- Inter-node Communication:
    - Utilizes publisher/subscriber model for asynchronous message passing
    - Service calls for synchronous interactions

Communication

- Topics:
  - Channels for message passing between nodes
  - Anonymous publish/subscribe mechanism
- Messages:
  - Typed data structure
  - Defined using a simple language (ROS msg)

Services and Actions

- Services:
  - Request/response interaction between nodes
  - Defined by a pair of message structures: a request and a response
- Actions:
  - Goal-oriented communication for longer-running tasks
  - Provides feedback, status, and results

Why Pub-Sub?

- Decoupling of Components:
  - Separates message production and consumption
- Flexibility and Scalability:
  - Easily incorporates additional nodes
  - Facilitates complex interactions among nodes
- Real-Time Data Handling:
  - Efficient handling of real-time data streams
- Ease of Integration:
  - Simplifies integration of diverse hardware and software

Sources

- https://vulcan.io/blog/how-to-secure-microservices-the-complete-guide/
- https://www.uber.com/en-CA/blog/cybersecurity-incident-simulation/?uclick_id=531d09e7-1373-4fe5-9f66-484d0714031c
- https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber
- https://arxiv.org/pdf/2211.07752.pdf
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204A.pdf
- https://www.codemag.com/Article/2203061/Secure-Microservices