



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Automating and Scaling of Threat Modelling for Cloud-Native Architecture

March 25th 2026, Security track



Introduction



KubeCon



CloudNativeCon

Europe 2026



Hanna Papirna

- Lead Security Engineer, Consultant
- CloudSec (Azure), IAM, threat modelling
- Expert in cloud secure landing zones adoption and security posture management
- Cloud Transformation advisory

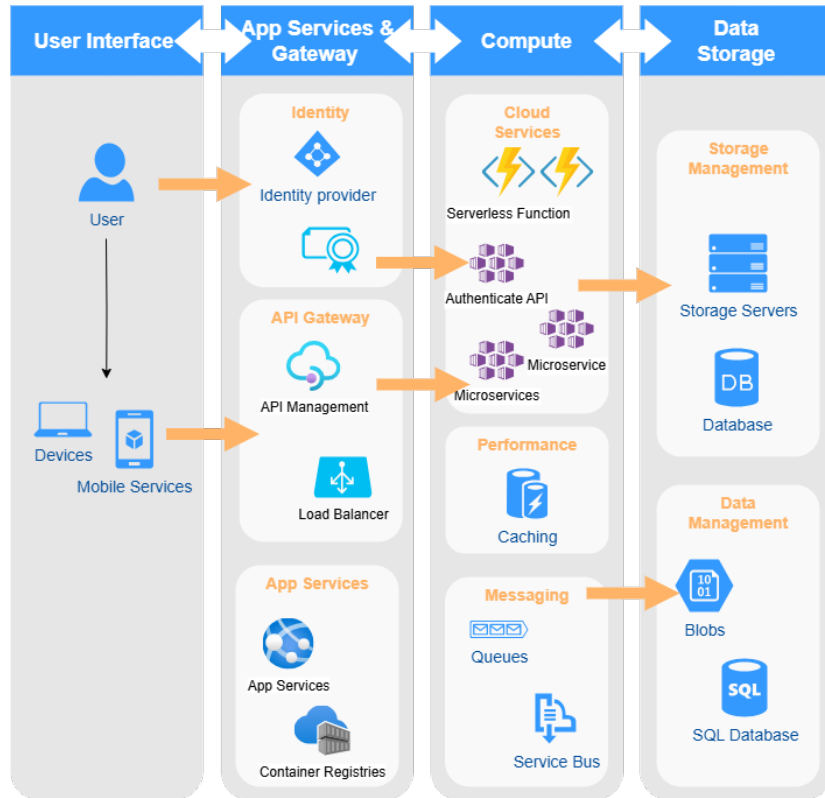


Emma Yuan Fang

- Senior Security Architect & COE Lead for UK&I
- AppSec, CloudSec (Azure), DevSecOps
- Mentor @ WiCyS, Cajigo, UK Universities
- Cloud Security Advocate, Advisory board member

Disclaimer: The views in this presentation are our own and not those of our employers or affiliates. The content is for informational purposes only and all ideas are open for further discussion.

The State of Cloud Threat Modelling



“76% of organizations now rely heavily on Cloud-Native development.”

- CNCF

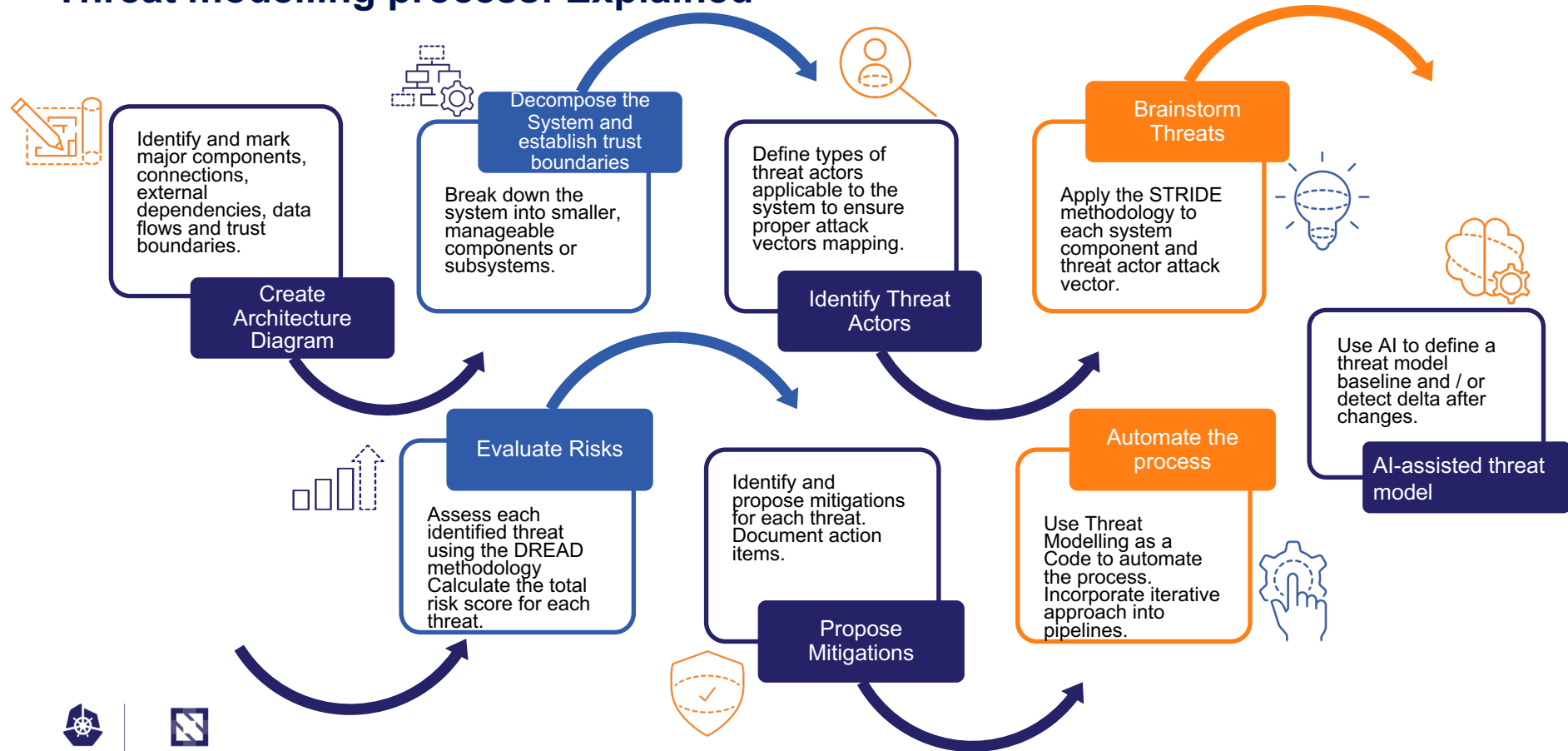
“by 2025, over 95% of new digital workloads will be deployed on cloud-native platforms.”

-Gartner

Cloud & Cloud-Native Threat Modeling Challenges

- Shared responsibility models
- Managed services and APIs
- Multi-tenant and identity federation considerations
- Dynamic infrastructure (IaC, serverless, containers)

Threat modelling process: Explained



Existing threat modelling frameworks

STRIDE – same old golden standard

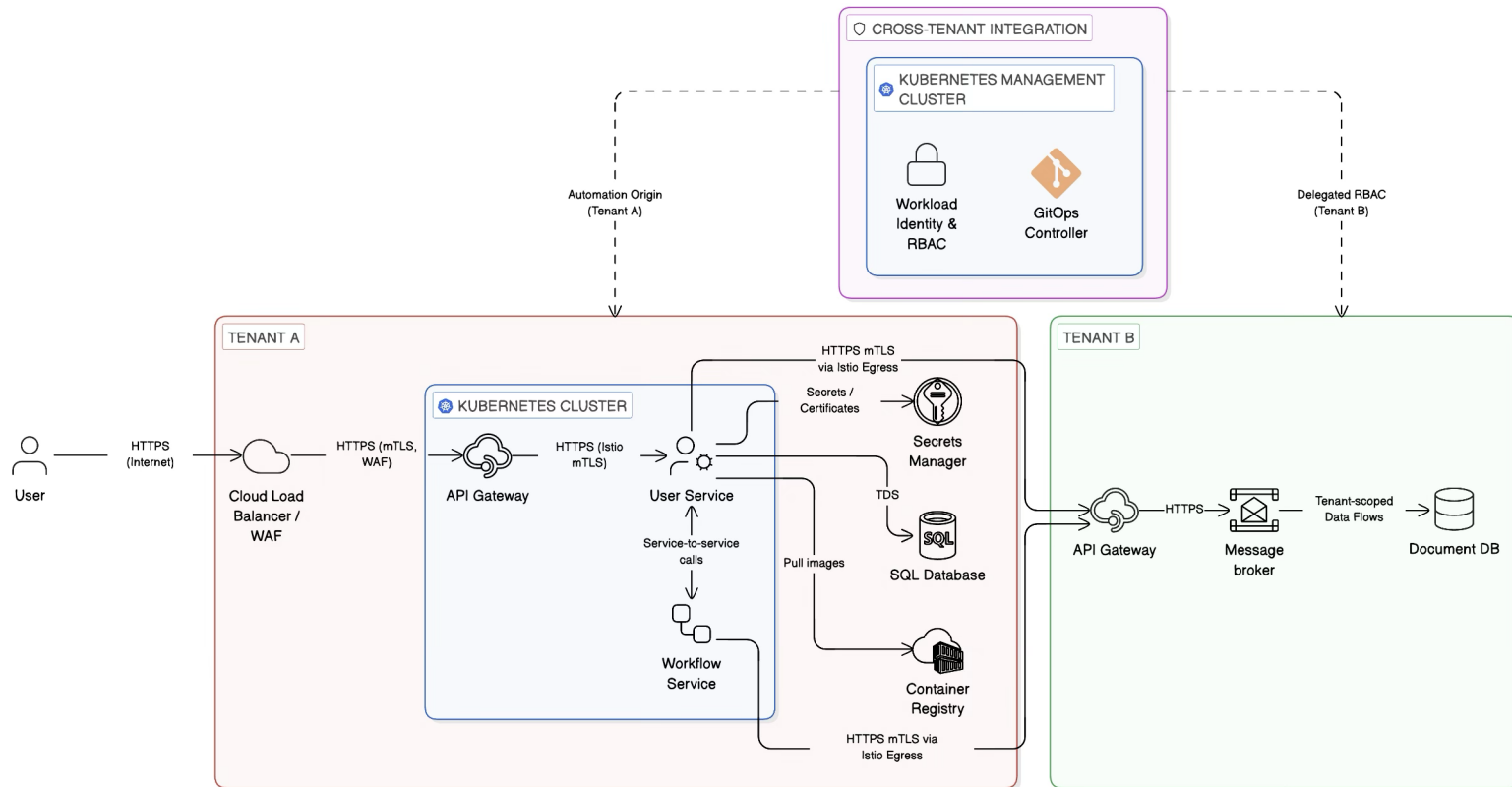
- **Why are we still using it today, even in Kubernetes?**
- Structured coverage, embedded in CNCF, monolithic checklist
- **Alternatives like VAST** (Visual, Agile, and Simple Threat modeling)
- Demand cultural shift to collaborative workshops, which regulated sectors resist due to audit traceability requirements
- **The CNCF Security Technical Advisory Group** provides the primary threat modelling guidance – lightweight, STRIDE-based assessments
- **Microsoft and CNCF** offer STRIDE adaptations for Kubernetes, emphasizing spoofing via service accounts, elevation via RBAC drift, and DoS
- **MITRE ATT&CK for Containers** further visualizes tactics like pod escapes and lateral movement across namespaces

Gaps in threat modelling frameworks

1. **Dynamic Complexity Overwhelm:** ephemeral pods, auto-scaling service meshes, and cross-tenant flows create shifting trust boundaries
2. **Cross-Tenant Blind Spots:** models excel for single-project isolation but miss multi-tenant delegation patterns
3. **Scalability Bottleneck:** manual assessments don't keep pace with CI/CD velocity
4. **Gap 4: Invisible Trust Boundaries:** monolithic models assume a single static perimeter, but in cross-tenant Kubernetes, trust boundaries are dynamic and multi-layered

Cross-tenant Identity Governance Platform

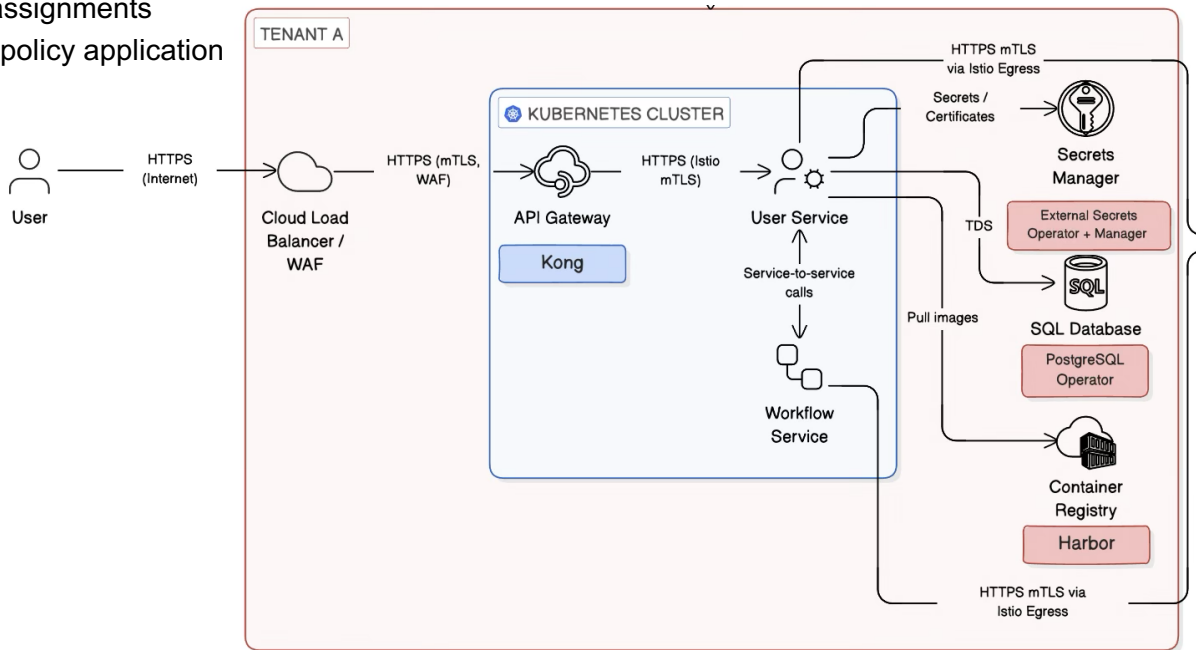
This solution is designed to centralize user lifecycle management. The platform ensures secure, automated identity operations – streamlining user onboarding, access revocations, and policy synchronization between distributed environments.



Tenant A (Managing Tenant)

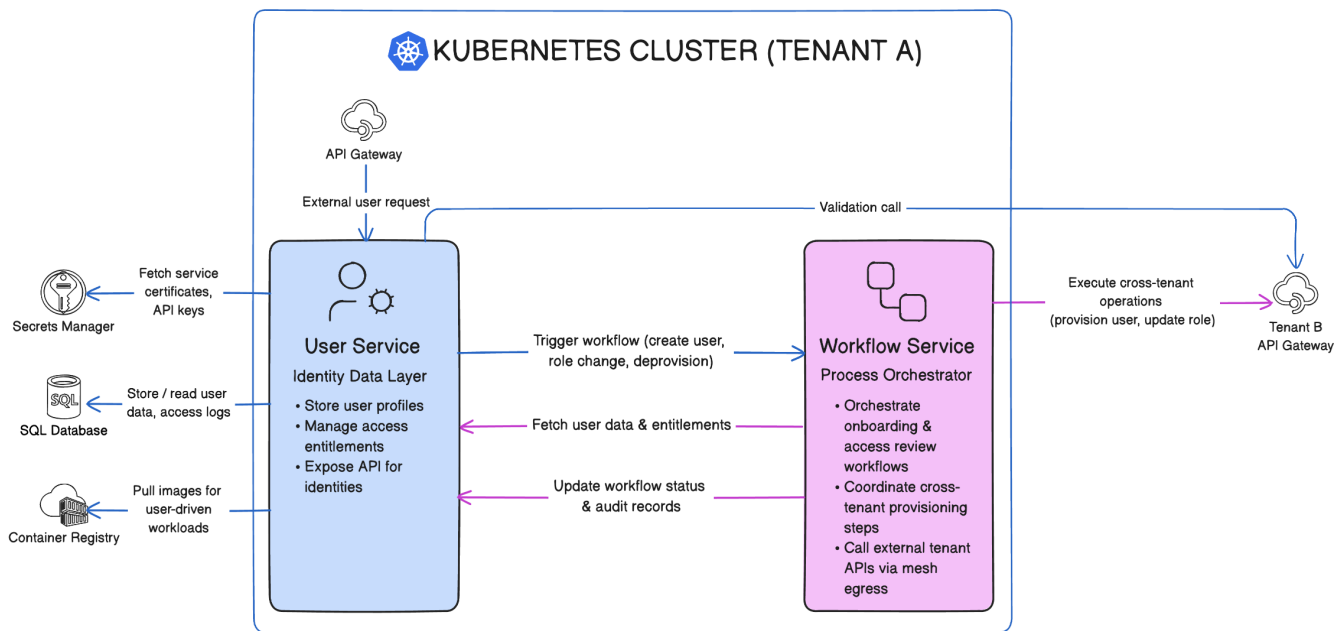
Key Processes/Functionalities:

- User onboarding, provisioning, and lifecycle updates
- Identity attribute synchronization and role assignments
- Initiates cross-tenant access requests and policy application
- Enforces compliance and audit workflows



Inside Kubernetes Cluster: User & Workflow Services

The Kubernetes cluster serves as the core orchestration and execution layer for the identity governance platform's microservices, delivering scalability and security through Istio service mesh with automatic mTLS encryption.



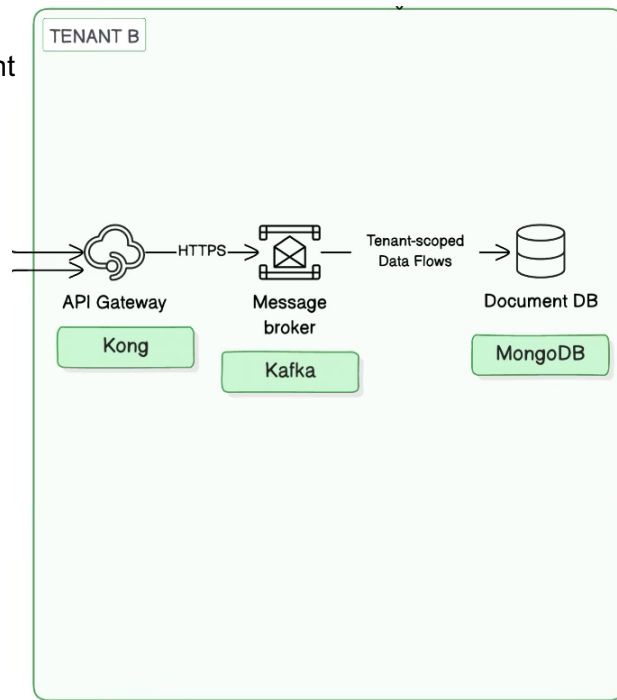
Cross-tenant Identity Governance Platform: Target Tenant

Tenant B (Target Tenant)

Primary Role: Receives delegated operations and enforces access/policy changes initiated by Tenant A.

Key Processes/Functionalities:

- Exposes APIs for user account synchronization and lifecycle management
- Validates and enforces cross-tenant access privileges and roles
- Processes automated remediation tasks and compliance checks from managing tenant
- Manages local data stores (e.g., business records, per-tenant compliance data)



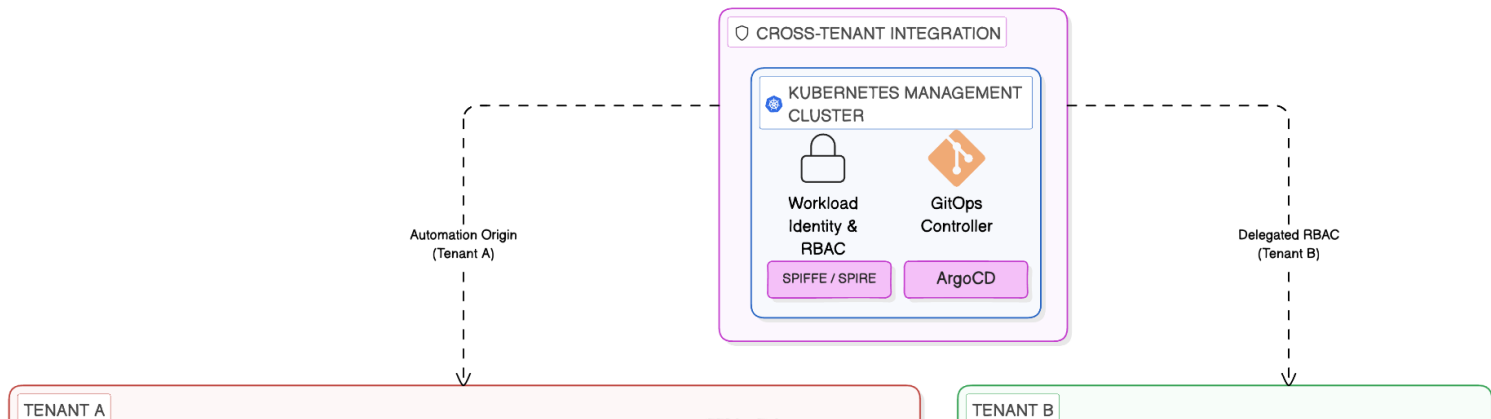
Cross-tenant Identity Governance Platform: Integration Component

Cross-tenant integration component

Primary Role: Enables secure, permissioned delegated management across tenants with granular role-based access control.

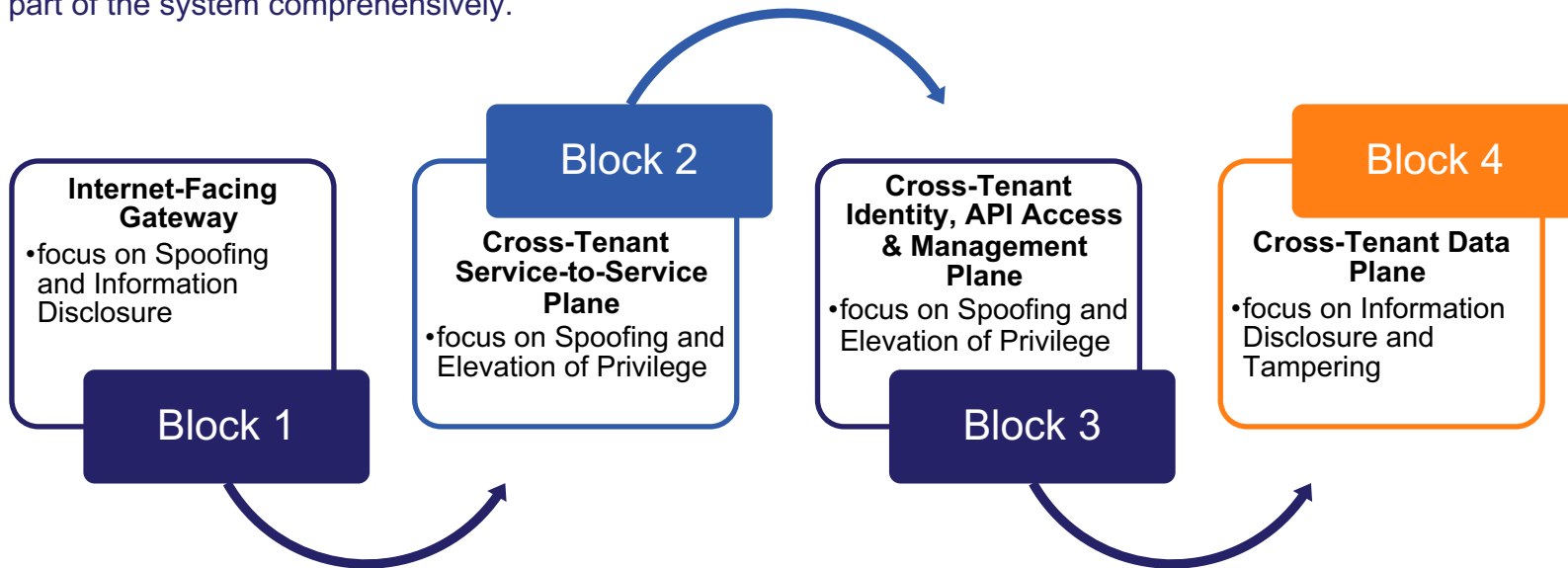
Key Processes/Functionalities:

- Grants Tenant A controlled access to specific resources in Tenant B via GitOps deployments and workload identities.
- Provides just-in-time, least-privilege access for cross-tenant operations through delegated RBAC pushed from the management plane
- Audits all cross-tenant management activities to support compliance and non-repudiation
- Facilitates large-scale, automated operations (automation origin to Tenant A, RBAC to Tenant B) without sacrificing tenant segregation



Decompose the System and Session

Break down the system into smaller, manageable components or subsystems. This makes it easier to apply STRIDE to each part of the system comprehensively.



Define a clear and efficient scope for each session block. Divide the sessions by major components and within each session, address a subset of STRIDE threats.

Example Block 1: Internet-Facing Gateway (Edge Ingress)

Traditional threat modelling may miss

- Treat internet-facing components as single-entry points with static perimeter controls. They fail to capture dynamic tenant routing complexity, where ingress routes requests to tenant-specific backends;
- Miss tenant enumeration where attackers probe hostnames/paths, and cross-tenant configuration bleeding where shared annotations affect traffic.

Components: Tenant A: Cloud Load Balancer / WAF

Trust Boundary: Internet → Tenant A edge / cluster boundary

Responsibility: Shared: platform team owns ingress policies; infra provider delivers network / load balancer services

CNCF Help & Gaps

- Gateway API and WAF provide strong foundations for route isolation and TLS termination;
- Configuration drift across tenants is not prevented automatically – teams must implement namespace-scoped policies and continuous validation.

Spoofing (S)

- **Cross-Tenant Hostname Impersonation:** Attackers register malicious hostnames mimicking tenant-specific routing to bypass ingress routing rules and inject forged requests into tenant backends
- **Ingress Controller Certificate Replay:** Compromised TLS certificates allow replay attacks at the edge, impersonating legitimate internet-to-cluster traffic

Information Disclosure (I)

- **Tenant Enumeration via Edge Diagnostics:** Logs, debug endpoints, and annotation misconfigurations expose tenant namespaces, internal hostnames, and backend services during probing
- **Routing Configuration Leakage:** Shared CRDs or diagnostic endpoints reveal dynamic tenant routing rules, enabling attackers to map backend services and craft targeted exploits



Example Block 2: Cross-Tenant Service-to-Service Plane (Service Mesh + East/West)

Traditional threat modelling may miss

- Treat internal communications as trusted single-hop and miss mTLS negotiation/service discovery;
- Fail to capture cross-tenant pod-to-pod threats, ephemeral identities, or cascade failures leveraging shared trust across tenants.

Components: Tenant A: User & Workflow microservices Kubernetes cluster (Istio service mesh)

Trust Boundary: Internal network → pod network → cross-tenant service communication

Responsibility: Customer-managed mesh + Kubernetes configuration (mTLS, authZ policies, service discovery)

CNCF Help & Gaps

- Istio's automatic mTLS between services encrypts all pod-to-pod traffic by default and validates service identities;
- Cannot prevent creating overly broad authorization policies or sharing trust domains across tenants.

Spoofing (S)

- **Istio mTLS SVID Theft:** Stolen SVIDs from pod memory or sidecar compromise allow impersonation of Kubernetes services, accessing cross-tenant paths despite mTLS enforcement
- **Service Account Token Projection Abuse:** Ephemeral Kubernetes service account tokens projected to Istio sidecars are harvested via pod escape, enabling spoofed east-west calls

Elevation of Privilege (E)

- **Istio AuthorizationPolicy Bypass:** Overly broad mesh policies allow compromised services to escalate access to cross-tenant endpoints
- **SPIFFE Trust Domain Overlap:** Shared mTLS trust domains between tenants enable identity chaining from low-privilege to high-privilege workloads



Example Block 3: Cross-Tenant Identity, API Access & Management Plane

Traditional threat modelling may miss

- Miss cross-tenant delegation patterns and policy inheritance;
- Fail to capture access policy drift, token loss, and attack surface from identity sync/delegated flows between tenants.

Components: Kubernetes management cluster (ArgoCD GitOps controller), Workload Identity & RBAC (SPIRE/SPIFFE), Tenant A: API Gateway Kubernetes cluster (Kong), Tenant B: API Gateway (Kong).

Trust Boundary: Tenant A ↔ Tenant B; workload identity issuance/attestation boundary

Responsibility: Shared across platform/security teams: policy delegation, identity federation, API gateway control-plane security, RBAC delegation model

CNCF Help & Gaps

- SPIRE workload identities eliminate shared secrets, ArgoCD enables GitOps workflows with audit trails, Kubernetes RBAC offers fine-grained access control;
- Over-permissioned roles or trust bundle misconfiguration across tenants are not prevented.

Spoofing (S)

- **SPIFFE Workload Identity Forgery:** Compromised SPIRE agents forge SVIDs, impersonating gateways to bypass RBAC delegation and access other tenant's APIs
- **GitOps Repo Compromise:** Stolen Git credentials allow attackers to impersonate legitimate app-of-apps deployments, pushing malicious manifests across tenants

Elevation of Privilege (E)

- **RBAC Delegation Drift:** Over-permissive SPIRE trust bundles or ArgoCD ClusterRoles enable low-priv Tenant A identities to escalate to Tenant B admin via chained delegation
- **API Gateway Policy Inheritance Abuse:** Shared policy CRDs allow attackers to elevate via mis-scoped authZ rules propagated from management cluster



Example Block 4: Cross-Tenant Data Plane

Traditional threat modelling may miss

- Treat storage as isolated and miss cross-tenant data flows, shared keys, multi-tenant queues;
- Fail to capture sovereignty violations and attack surface from shared topics/registries.

Components: Tenant A: Secrets manager (ESO + Manager), SQL database (PostgreSQL operator), Container registry (Harbor); Tenant B: Message broker (Kafka), Document DB (MongoDB)

Trust Boundary: Application services → data systems (events, secrets, artifacts)

Responsibility: Customer-managed (operators, ESO, Harbor, Kafka): + governance (isolation at schema/topic/ACL level, encryption and key separation)

CNCF Help & Gaps

- ESO automates secret synchronization, Harbor provides signed container images with vulnerability scanning, Kubernetes operators bring database lifecycle management;
- None prevent secrets from being over-shared across namespaces, or Kafka topics from lacking proper ACLs.

Information Disclosure (I)

- **Secret Rotation Bypass:** ESO + ExternalSecrets Operator misconfigurations leak secrets from one tenant to another via shared Kubernetes Secrets without proper RBAC isolation
- **Image Metadata Exposure:** Unencrypted Harbor tags/labels reveal supply-chain artifacts (e.g., vulnerable images tagged for cross-tenant use) via public registry introspection

Tampering (T)

- **Topic Poisoning:** Compromised Tenant A producers inject malformed events into shared Kafka topics, corrupting Tenant B MongoDB writes and workflow state
- **Database Operator Schema Drift:** Malicious operator reconciliations merge Tenant A schemas, tampering with row-level isolation and enabling cross-tenant data corruption



Know Your Adversaries

Before mapping threats, establish WHO is targeting your platform — motivation and capability shape which layers matter most.

External Attacker

- Nation-state APT groups & advanced persistent threats
- Organised cybercrime syndicates (ransomware, data brokers)
- Opportunistic threat actors exploiting public misconfigurations

Motivated by espionage, financial gain, or disruption. High capability and persistence, they probe edge, APIs, and exposed cloud surfaces repeatedly until a foothold is found.

► **Edge ingress** **Exposed APIs** **Cloud console** **Public-facing endpoints**

Malicious or Unintentional Insider

- Disgruntled engineer or departing admin abusing access
- Negligent developer with overly-permissive kubectl or IAM roles
- Accidental misconfiguration — open RBAC, exposed secrets, weak pod spec

Malicious insiders act with intent; unintentional insiders cause equal damage through neglect, drift, or misconfiguration left undetected.

► **kubectl / IAM roles** **CI/CD credentials** **RBAC policies** **Secrets & config**

Supply Chain Actor

- Compromised OSS maintainer injecting malicious code upstream
- Malicious or typosquatted container image in public registry
- Tampered Helm chart or GitOps manifest in a shared pipeline

Pre-plants the attack at build time, no runtime presence required. By the time the workload is running, the compromise is already trusted and signed in.

► **Container images** **Helm charts** **OSS dependencies** **GitOps manifests**

Automated / Bot Threats

- Cryptomining botnets targeting exposed compute and metadata APIs
- Automated vulnerability scanners probing known K8s misconfigs
- Credential-stuffing bots against API gateway and auth endpoints

Indiscriminate, high-volume, and fast. Exploits well-known misconfigurations and default credentials programmatically - no human operator required.

► **Metadata API** **Exposed dashboards** **Auth endpoints** **Compute resources**



KubeCon



CloudNativeCon

Europe 2026

Threat Actor Mapping Across All Attack Planes

BLOCK	THREAT ACTOR	ATTACK VECTOR	THREAT EXAMPLE	STRIDE	MITRE ATT&CK
Internet-Facing Gateway	External	Exposed Envoy ingress; unvalidated hostname routing	Cross-tenant hostname impersonation; cert replay	Spoofing	Initial Access
	Automated	Unauthenticated diagnostic & metrics endpoints	Tenant enumeration via edge diagnostics	Info Disclosure	Discovery
	Internal	Kong misconfigured policy scoping	API gateway policy inheritance abuse	Escalation	Defence Evasion
	Supply Chain	Compromised Envoy/Kong image in Harbor	Malicious proxy intercepts all tenant TLS traffic	Tampering	Collection
Service-to-Service Plane	External	Exploited microservice pivot into the mesh	Stolen SVID used to impersonate trusted service	Escalation	Defence Evasion
	Automated	Weakly scoped service calls; no mTLS enforcement	Shared trust domain exploited; credential harvest at scale	Spoofing	Credential Access
	Internal	Misconfigured pod spec: privileged: true or hostPID: true	Privileged container escape to host; cross-tenant workloads	Escalation	Priv. Escalation
	Supply Chain	Unverified image pulled without Cosign/Sigstore signing	Backdoored image runs reverse shell; exfiltrates pod secrets	Tampering	Execution
Management Plane	Internal	Ungoverned RBAC role bindings across tenant namespaces	RBAC delegation drift -> escalation to cluster	Escalation	Priv. Escalation
	Supply Chain	Compromised GitOps repo pushing malicious manifests	GitOps repo compromise -> stolen git credentials -> push malicious manifest	Tampering	Execution
	External	Exposed Kubernetes API server or workload identity endpoint	SPIFFE workload identity forgery; trusted impersonation	Spoofing	Credential Access
	Automated	Automated scanning of management cluster API surface	Credential harvesting; service account enumeration	Info Disclosure	Discovery
Tenant Data Plane	External	Unauthenticated or weakly authenticated Kafka topics	Cross-tenant message interception; event stream read	Info Disclosure	Collection
	Automated	Unscoped Kafka consumer group configuration	Tenant data enumeration via misconfigured topic ACLs	Info Disclosure	Discovery
	Internal	Direct MongoDB access with overly permissive service acct	Tenant data exfiltration bypassing app-layer controls	Info Disclosure	Exfiltration
	Supply Chain	Compromised Kafka or MongoDB operator image	Malicious operator exfiltrates all tenant data at rest	T — Tampering	Persistence

KEY PATTERN Every plane shares the same 4 threat actor archetypes — External Attacker, Automated/Bot, Malicious Insider, Supply Chain Actor — but each plane exposes a different dominant risk: **Spoofing at Internet-facing gateway**, **Privilege escalation in the service-service plane**, **Credential threats at management plane**, **Data exfiltration at the data plane**.



Evaluate Risks: DREAD Heat Map

Map DREAD scores to Likelihood and Impact, then plot on the risk heat map for a justified, structured risk rating.

DREAD FACTORS



Damage Potential

How bad would an attack be?



Reproducibility

How easy to reproduce the attack?



Exploitability

How much work to launch the attack?



Affected Users

How many people will be impacted?



Discoverability

How easy is it to discover threat?

RISK HEAT MAP

	Low Risk (1-2)	Medium Risk (3-4)	High Risk (5-6)
	Low Impact (1-2)	Medium Impact (3-4)	High Impact (5-6)
Low Likelihood (1-2)	Low Risk	Low Risk	Medium Risk
Medium Likelihood (3-4)	Low Risk	Medium Risk	High Risk
High Likelihood (5-6)	Medium Risk	High Risk	High Risk

FORMULA

$$\text{Likelihood} = (R + E + D) / 3$$

$$\text{Impact} = (D + A) / 2$$

Scores range 1 (Low) → 6 (High)

MAPPING DREAD TO RISK DIMENSIONS

LIKELIHOOD

$R + E + D$ — Reproducibility + Exploitability + Discoverability

IMPACT

$D + A$ — Damage Potential + Affected Users



Risk Evaluation and Mitigation

THREAT

GitOps Repo Compromise: Stolen Git credentials allow attackers to impersonate legitimate app-of-apps deployments, pushing malicious manifests across tenants

DREAD SCORING

DREAD Factor	Rating	Justification
Damage Potential (D)	High (5)	Blast radius is bounded by the stolen credential's repo permissions, branch protection and mandatory PR review prevent direct reconciliation without human approval.
Affected Users (A)	High (5)	The app-of-apps pattern by design governs all tenant application trees from a single repo, a malicious manifest pushed here propagates to every tenant simultaneously with no additional effort from the attacker.
Reproducibility (R)	Medium (4)	Once Git credentials are stolen, pushing a malicious manifest is as repeatable and reliable as a normal developer commit, the GitOps controller will reconcile it automatically on every sync loop.
Exploitability (E)	Medium (3)	Stealing Git credentials requires social engineering, phishing, or secrets leakage from CI/CD logs, moderately accessible to an insider or sophisticated external attacker.
Discoverability (D)	Medium (3)	Git credential exposure is frequently discoverable via secrets scanning of CI/CD pipeline logs, .env files, or GitHub Actions environment variables — a well-known and actively hunted attack surface.

MITIGATION:

Enforce OIDC-federated short-lived Git credentials, require signed commits via branch protection rules so unsigned pushes are rejected before the GitOps controller reconciles them. Mandate two-reviewer PR approval on the app-of-apps repo and gate ArgoCD/Flux sync with a OPA policy check that validates tenant scoping on all manifests before reconciliation.

RISK RESULT**LIKELIHOOD**

$$(R(4) + E(3) + D(3)) / 3$$

3**Medium****IMPACT**

$$(D(5) + A(5)) / 2$$

5**High****RISK LEVEL****HIGH**

Medium Likelihood (3) × High Impact (5)

→ **HIGH RISK** (per heat map)



KubeCon

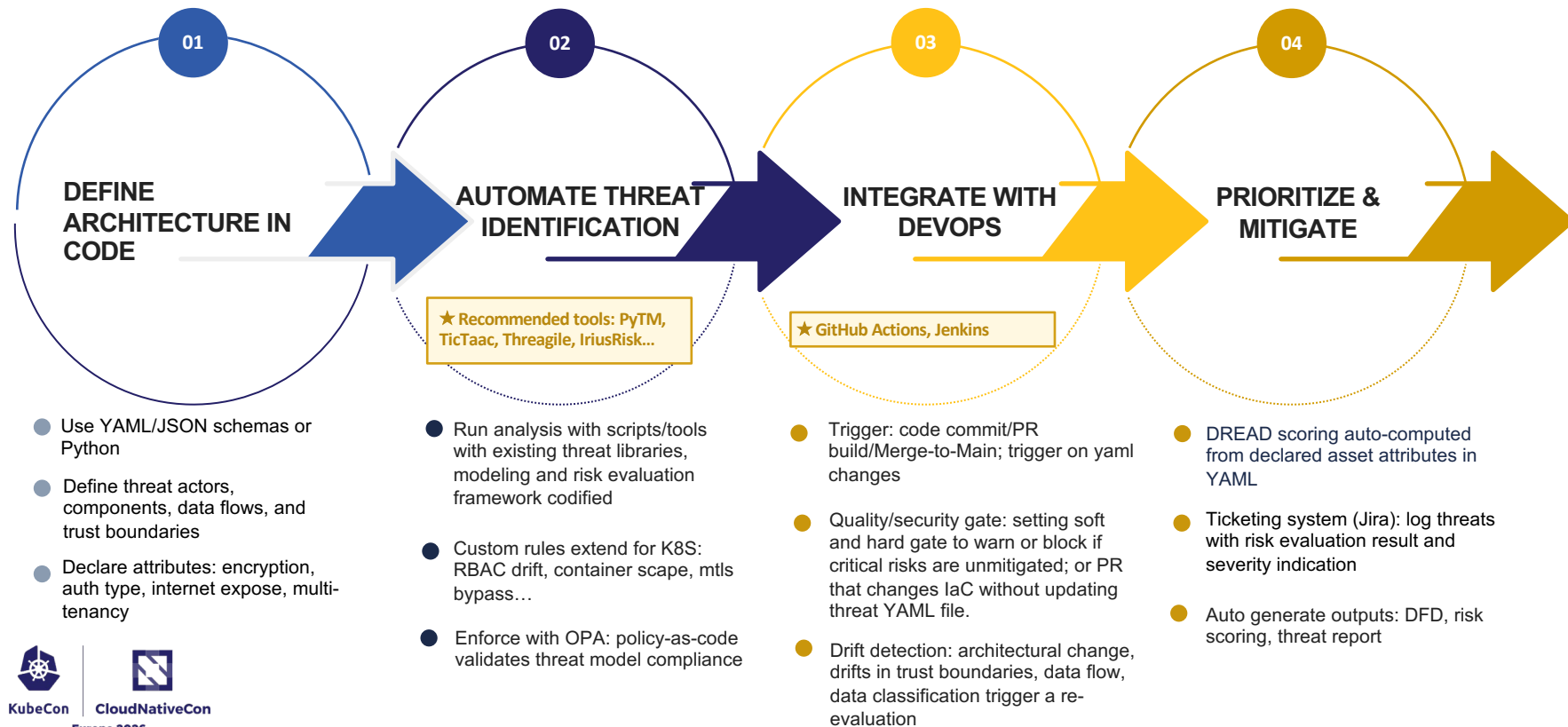


CloudNativeCon

Europe 2026

Scaling and Automation with TMasC

General approach to Threat Modelling as Code



What Can Be Automated in Practice?

What TMaaC and DevOps automation cannot replace (based on threat modeling manifesto)

AUTO — TMasC + CI/CD Pipeline	Manual — Expert Judgment
System Decomposition	
✓ Parse IaC / YAML to extract all components ✓ Auto-generate DFD on every pipeline run ✓ Detect trust boundary crossings in data flows	- Define system purpose and business intent - Set data sensitivity classifications (PII, etc.) - Capture undocumented / implicit assumptions
Threat Identification	
✓ Apply STRIDE rule library against model graph ✓ Map findings to MITRE ATT&CK, CWE, CAPEC ✓ Evaluate findings risk level according to risk framework (e.g. DREAD)	- Identify novel & chained multi-step attack paths - Reason about attacker motivation & business logic - Assess insider threats and social engineering risk
Risk Evaluation	
✓ Suggest standard mitigations from built-in library ✓ Auto-create Jira tickets for identified risks ✓ Block pipeline merge on unmitigated critical risks	- Decide: accept / mitigate / transfer each risk - Design custom controls for novel threats - Balance remediation priority vs. sprint capacity
Validation and Mitigation	
✓ Verify all declared risks have a recorded status ✓ Generate versioned audit trail via Git history ✓ Regression scan on every pipeline run	- Security team to validate controls work - Sign-off on residual risk - Update model retrospectively after incidents

KEY PRINCIPLE Automation owns consistency, speed, and known-pattern coverage. Humans own completeness, novelty, and judgment. Tools enhance the human process; they do not replace it.

TMasC in Practice: Input

Example:Threagile

Architectural definition in YAML input model (actors, components, boundaries, flows)

```
# — DATA ASSETS —————
data_assets:
  user-identity-data:
    id: user-identity-data
    description: "User profiles, roles, entitlements (PII/PCI)"
    usage: business
    origin: customer
    quantity: many
    confidentiality: strictly-confidential # PII + entitlements
    integrity: critical
    availability: critical

  workflow-audit-records:
    confidentiality: confidential
    integrity: critical # immutable audit trail

# — TRUST BOUNDARIES —————
trust_boundaries:
  kubernetes-cluster-tenant-a:
    type: kubernetes-namespace
    description: "Tenant A AKS - Istio mTLS mesh"
    technical_assets_inside:
      - user-service
      - workflow-service

  tenant-b-boundary:
    type: cloud-provider-account # separate AAD tenant
    technical_assets_inside:
      - tenant-b-api-gateway
```

```
# — TECHNICAL ASSETS —————
technical_assets:
  user-service:
    type: process
    technology: service-mesh
    trust_boundary: kubernetes-cluster-tenant-a
    machine: container
    confidentiality: strictly-confidential
    internet: false # internal only via Istio
    multi_tenant: true # cross-tenant calls
    data_assets_processed:
      - user-identity-data
      - workflow-audit-records

  workflow-service:
    type: process
    technology: service-mesh
    trust_boundary: kubernetes-cluster-tenant-a
    multi_tenant: true # orchestrates cross-tenant steps

  tenant-b-api-gateway:
    type: process
    trust_boundary: tenant-b-boundary
    internet: true # cross-tenant egress target

# — DATA FLOWS —————
communication_links:
  user-svc-to-workflow-svc:
    source: user-service # trigger workflow
    target: workflow-service
    protocol: https
    authentication: service-account # KSA token
    encrypted: true # Istio mTLS

  workflow-svc-to-tenant-b:
    source: workflow-service
    target: tenant-b-api-gateway
    crosses_trust_boundary: true # cross-tenant
    protocol: https
    authentication: service-account # workload identity federated
    data_sent: [ user-identity-data ] # PII cross-boundary!

  user-svc-to-tenant-b-validation:
    source: user-service
    target: tenant-b-api-gateway
    crosses_trust_boundary: true
    protocol: https # validation calls
```



KubeCon



CloudNativeCon

Europe 2026

TMasC in Practice: Output

Example:Threagile - Threat outputs with risk status, severity, likelihood, and other component related information.

```
[
  {
    "category": "unnecessary-data-transfer",
    "risk_status": "unchecked",
    "severity": "medium",
    "exploitation_likelihood": "unlikely",
    "exploitation_impact": "medium",
    "title": "\u003cb\u003eUnnecessary Data Transfer\u003c/b\u003e of \u003cb\u003eService Certs\u003c/b\u003e",
    "synthetic_id": "unnecessary-data-transfer@svc-certs@usr-svc@vault",
    "most_relevant_data_asset": "svc-certs",
    "most_relevant_technical_asset": "usr-svc",
    "most_relevant_trust_boundary": "",
    "most_relevant_shared_runtime": "",
    "most_relevant_communication_link": "",
    "data_breach_probability": "improbable",
    "data_breach_technical_assets": [
      "usr-svc"
    ]
  },
  {
    "category": "missing-identity-store",
    "risk_status": "unchecked",
    "severity": "medium",
    "exploitation_likelihood": "unlikely",
    "exploitation_impact": "medium",
    "title": "\u003cb\u003eMissing Identity Store\u003c/b\u003e in the threat model (referencing \u003cb\u003eIngress Gateway\u003c/b\u003e)",
    "synthetic_id": "missing-identity-store@ingress-gw",
    "most_relevant_data_asset": "",
    "most_relevant_technical_asset": "ingress-gw",
    "most_relevant_trust_boundary": "",
    "most_relevant_shared_runtime": "",
    "most_relevant_communication_link": "",
    "data_breach_probability": "improbable",
    "data_breach_technical_assets": []
  }
],
```

```
{
  "category": "unguarded-access-from-internet",
  "risk_status": "unchecked",
  "severity": "elevated",
  "exploitation_likelihood": "very-likely",
  "exploitation_impact": "medium",
  "title": "\u003cb\u003eUnguarded Access from Internet\u003c/b\u003e of \u003cb\u003eUser Service Certs\u003c/b\u003e",
  "synthetic_id": "unguarded-access-from-internet@usr-svc@ingress-gw@ingress-gw\u003c/b\u003e",
  "most_relevant_data_asset": "",
  "most_relevant_technical_asset": "usr-svc",
  "most_relevant_trust_boundary": "",
  "most_relevant_shared_runtime": "",
  "most_relevant_communication_link": "ingress-gw\u003c/b\u003e",
  "data_breach_probability": "possible",
  "data_breach_technical_assets": [
    "usr-svc"
  ]
}
```

```
{
  "category": "missing-network-segmentation",
  "risk_status": "unchecked",
  "severity": "medium",
  "exploitation_likelihood": "unlikely",
  "exploitation_impact": "medium",
  "title": "\u003cb\u003eMissing Network Segmentation\u003c/b\u003e to further encapsulate and protect \u003cb\u003eDatabase\u003c/b\u003e",
  "synthetic_id": "missing-network-segmentation@db",
  "most_relevant_data_asset": "",
  "most_relevant_technical_asset": "db",
  "most_relevant_trust_boundary": "",
  "most_relevant_shared_runtime": "",
  "most_relevant_communication_link": "",
  "data_breach_probability": "improbable",
  "data_breach_technical_assets": [
    "db"
  ]
},
```





KubeCon



CloudNativeCon

Europe 2026

AI-Assisted Threat Modelling

STRIDE GPT tool: <https://tinyurl.com/4xr4j3vy>

Temperature and top_p enhancement: <https://tinyurl.com/264jx6vx>



AI Threat Models: Generic vs. Block-Aware Output

Threat	Threat Model 1 (more generic)	Threat Model 2 (more precise)	Difference
SPIFFE / SPIRE Identity Spoofing	- Misconfigured Istio mesh component bypasses mTLS or SPIFFE validation - Attacker compromises a Kubernetes service account or SPIFFE identity - Impersonates User/Workflow Service to make unauthorized cross-tenant calls to Tenant B API Gateway	- HOW: Compromises SPIRE agent in Tenant A - MECHANISM: Exploits agent to forge SVID mimicking Tenant A API Gateway - IMPACT: Uses forged gateway identity to bypass delegated RBAC and cross-tenant policies at Tenant B API Gateway	TM1 describes generic "bypass mTLS" without specifying where or how. TM2 pinpoints the exact attack vector (SPIRE agent compromise), what identity is forged (API Gateway), and which security boundary breaks (delegated RBAC + cross-tenant policies).
GitOps / ArgoCD Compromise	- Attacker compromises the Git repository used by ArgoCD - Introduces malicious configuration or policy changes - ArgoCD automatically applies them to Tenant B clusters - Leads to unauthorized access or service disruption	- HOW: Tampered with Git repo or CI/CD pipeline (upstream dependency hijacking, stolen credentials, poisoned Helm charts) - MECHANISM: ArgoCD deploys malicious manifests using its service accounts + SPIFFE identities - IMPACT: Injects over-permissive ClusterRoles, fake SPIRE workloads, backdoored Kong/Envoy policies → enables RBAC delegation drift to Tenant B cluster-admin	TM1 gives basic "bad Git → bad config" flow. TM2 details the full supply-chain attack: how GitOps becomes a trusted automation vector, what gets injected (SPIRE + RBAC + gateway policies), and the cross-tenant escalation path. Treats GitOps as an attack surface, not just automation.



AI-Assisted Threat Modeling: what works

What AI-assisted threat modelling does exceptionally well:

Historical Incident Pattern Recognition

- AI links architecture patterns to public breach post-mortems and CVE exploitation chains in similar cloud-native environments.
- Provides real-world context for threat prioritization (hypothetical vs. actively exploited).
- Connects abstract STRIDE categories to concrete attacker techniques documented in breach reports.

Converting security into developer action items

- AI transforms abstract threat descriptions into concrete Kubernetes/CNCF remediation steps that platform engineers can implement.
- Bridges gap between security team (threat language) and platform team (K8s primitives).
- Turns threat models into GitOps PRs teams can review.

Simulating "what if" architecture changes

- AI can model impact of architectural changes on threat surface before you make them.
- AI runs delta analysis: threat surface before/after the change.
- Enables security-aware architecture decisions in sprint planning.

AI-Assisted Threat Modeling: limitations

Critical pitfalls of AI-assisted threat modelling:

Generic threat trap

- AI models trained on monolithic apps generate OWASP/STRIDE checklists but miss CNCF-specific threats.
- Training data lacks cloud-native multi-tenant architectures. Models don't understand K8s primitives.
- Modern OWASP/STRIDE handle K8s (but LLMs don't automatically). LLMs can use them if prompted explicitly. Without that, they default to generic web app threats.

Ignoring existing mitigations

- AI tools analyze systems in isolation, failing to account for existing security controls and cloud-native protective mechanisms already implemented in the architecture.
- AI analyzes architecture descriptions, not live cluster state. No integration with K8s API, service mesh configs, or policies.
- Security teams waste time implementing redundant controls or ignoring valid AI output because it's full of noise.

Missing multi-hop cross-tenant attack paths

- AI treats each component in isolation – can't reason across chained exploits.
- Current models lack graph reasoning over K8s object relationships (service accounts → RBAC → pods → services → ingress → external APIs).
- Miss the exact attack chains adversaries use: lateral movement through service mesh, privilege escalation via workload identities, cross-tenant pivots.

Making AI-Assisted Threat Modeling Work

Practical Solutions:

- **Iterative refinement:** apply block decomposition methodology we discussed previously, using AI to analyse each security domain separately rather than the entire system at once
- **Model provider choice matters:** reasoning tokens let the model internally plan multi-step checks - e.g., "Does any path cross a tenant boundary without mTLS?" - before emitting final threats (e.g. Claude Opus 4.6, Claude Sonnet 4.6, OpenAI o3, Gemini 3.1 Pro etc.). Reasoning tokens deliver deeper causal chains and context-aware ranking
- **Temperature calibration for stability:** lower temperature produces more deterministic, repeatable threat lists
- **Fixed top_p value for consistent sampling:** nucleus sampling limits token choices to the most probable p% of options
- **Repository versioning as source of truth:** curated threat model in Git not the raw AI output. Run AI as a brainstorming delta against the versioned baseline
- **Result caching for incremental updates:** cache the last accepted threat model for each system as a baseline. Pass current model + delta to the AI in "update mode"



KubeCon



CloudNativeCon

Europe 2026

“Threat modelling isn’t just a box to check, it’s about understanding your unique cloud architecture, thinking like an attacker, and designing systems that are resilient from day one.”

What is the Future of Cloud-Native Threat Modelling?



Hanna Papirna

Lead Security Engineer | Cloud Security ☁️



Emma Fang, CISSP

Senior Security Architect @ EPAM
| Cloud Security, Strategy | Cloud Sec...



Thank you!



KubeCon



CloudNativeCon

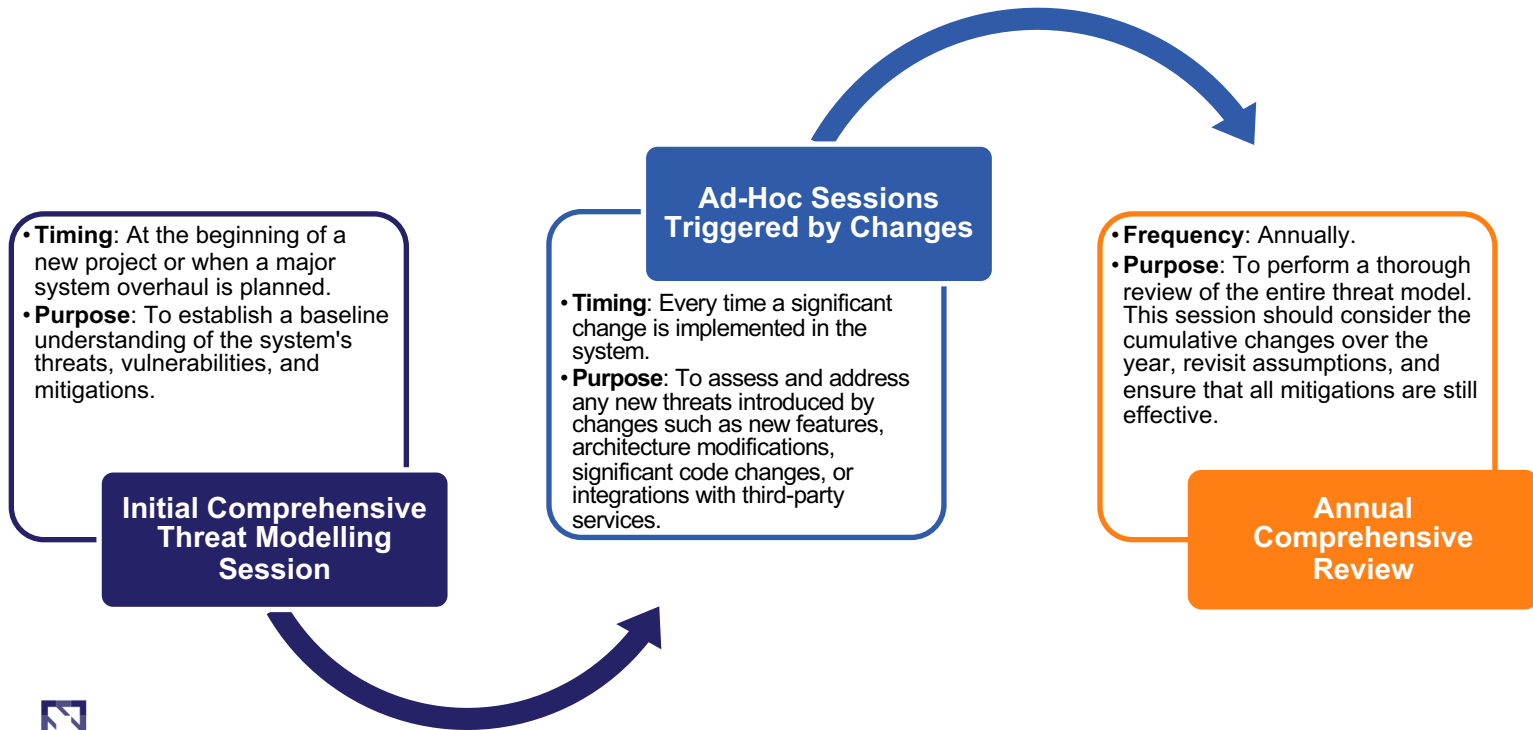
Europe 2026

Bonus content



Threat modelling sessions schedule

Regular threat modelling sessions are crucial for maintaining a robust security posture, especially in dynamic environments where systems and threats evolve continuously. The frequency of these sessions should balance thoroughness with practicality, ensuring that they are regular enough to keep up with changes but not so frequent that they become burdensome or redundant.



When to conduct initial (pre-production) session?

Initial System Architecture and Design

high-level architecture, including main components, data flows, and initial security design, are defined but not yet finalized.

Initial Threat Modelling Session

Detailed Design and Implementation

- Initial design is ready, and coding begins.

Pre-Production Finalization

- The system is fully finalized for deployment to the pre-production environment.

Optimal timing: The initial threat modelling session should be conducted **after the initial system architecture and (security) design are completed but before detailed design and implementation begin**. This ensures that security considerations are integrated early, and any identified threats can be addressed in the detailed design and implementation phases.

Security design: The first version of security design should be completed **before the initial threat modelling session**. This will allow the team to analyze and evaluate security measures already proposed. This helps in identifying any gaps or weaknesses in the design and ensures the security considerations are integrated early.



KubeCon

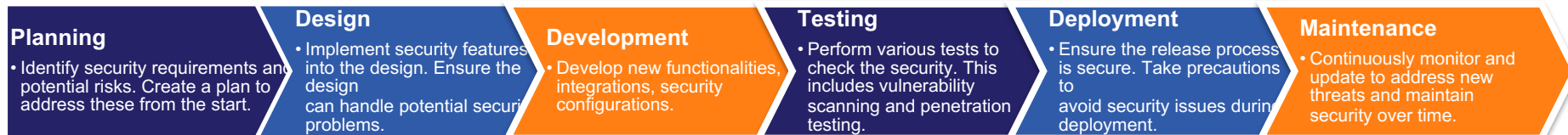
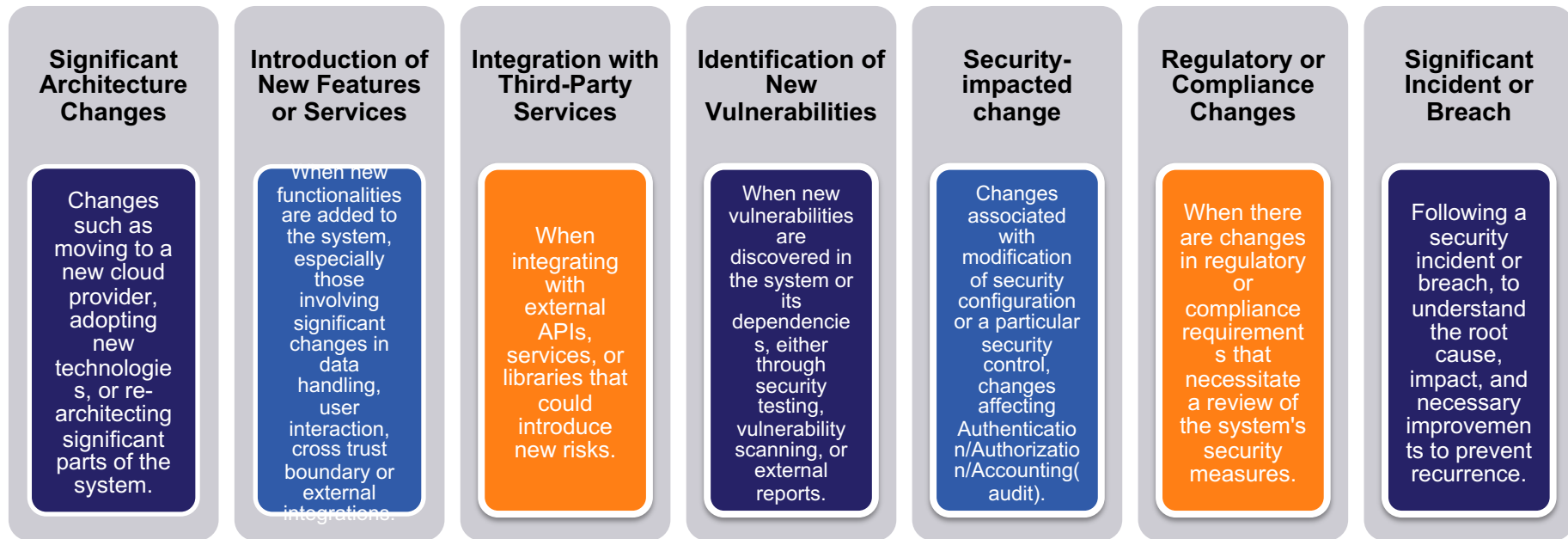


CloudNativeCon

Europe 2026

Triggers for Ad-Hoc Threat Modelling Sessions

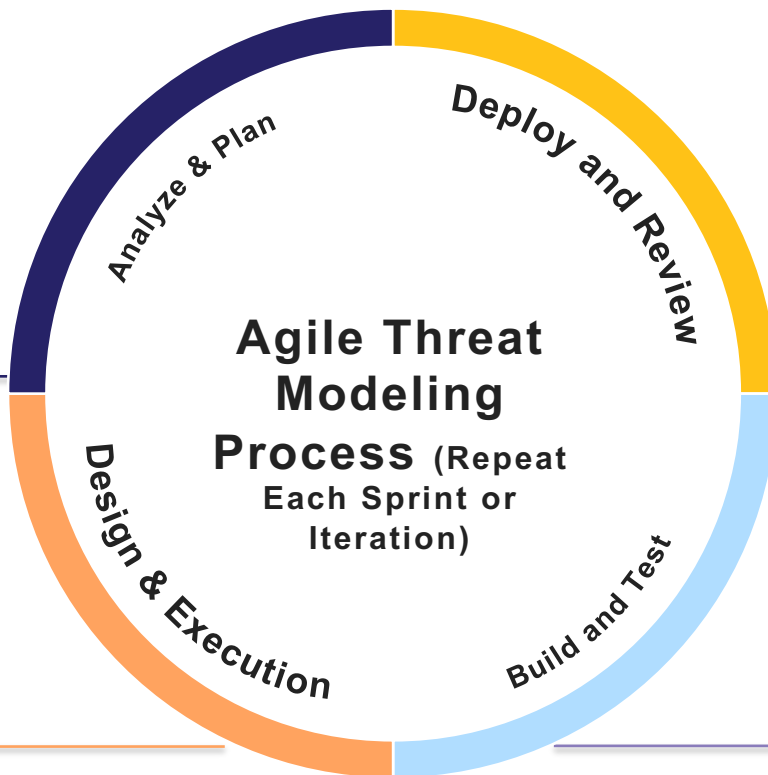
Example of events that can trigger an ad-hoc threat modelling session to ensure that new or modified threats are promptly identified and addressed with connection to the Secure Software Development Lifecycle.



Stakeholder Engagement: The Right People at the Right Time

- **Stakeholders:** PM, DM, BA, Architects, Security Lead
- Plan threat model activities for the sprint
- Create user stories and security stories

- **Stakeholders:** Architects, Developers, Security Lead
- Conduct sprint-based threat modelling, focus on specific microservice or feature
- Create user stories and security stories
- Design refinement



- **Stakeholders:** DM, Platform team, QA Tester
- Mitigation & remediation consulting
- Conduct a retrospective to identify lessons learned and provide feedback
- Update threat models and security documentation.

- **Stakeholders:** Security Engineer, Security Tester, QA Engineer/Manager, Developers
- Implement Security Control
- Conduct code reviews with a focus on security.
- Validate threat model with automated security testing



Manage Your Threats

Identify and propose mitigations for each identified threat. Document the proposed security controls and countermeasures.

The output from the mitigation sessions should contain a list of proposed security measures and then be moved from whiteboard to ticketing system/team's backlog (e.g Jira) to implement security enhancements based on the priorities and defined SLAs.

The ticket can provide generic explanation and can be further detailed during regular refinement meetings.

It will be beneficial to mark the tickets with necessary tags, based on the type of threats and level of risks to provide priority clarification.