
CLOUDNATIVE
SECURITYCON
NORTH AMERICA 2023

CLOUDNATIVE
SECURITYCON
NORTH AMERICA 2023

Policy Governance with OSCAL

Robert Ficcaglia

$id

Robert Ficcaglia, co-chair wg-policy

$ ls -al .stuff_i_do

- Kubernetes sig-security
- SOX, SOC2, HIPAA, ISO, HITRUST
- FISMA, CJIS, FedRAMP
- OSCAL
- Machine Learning…especially with Graphs

$ history

- Focused on Kubernetes (P)olicy and (p)olicy
- Contributions to Kubernetes community
  - PolicyReport CRD (aligned with OSCAL)
  - Kubernetes Policy White Paper
  - PolicyReport Dashboard + adapters (Falco, Kubeconf, Kyverno, more)
- In progress
  - Policy Governance Whitepaper
  - OSCAL Examples and NIST collaboration
  - Review and use cases for CEL based Admission Control
  - PolicyReport KEP

Basic (P)olicy Questions

- Who should have access to IaC/clusters?
- Who can deploy workloads and services to your clusters?
- What resources should be in my IaC/clusters?
- What actions can users perform?
- What permissions do workloads have within clusters?
- What are the network policies between workloads within clusters?

Configuration (p)olicy Questions

- What resource configs can be deployed in your clusters?
- How to isolate workloads?
- How to manage data flows?
- How do you set limits on resources available to a workload?
- What baseline configuration standards/defaults for workloads?
- Can I verify the integrity of workload images?
- Cross cutting concerns - eg. storage *and* access roles?
- Provisioning or meta-admin cluster(s)?

CONOPS

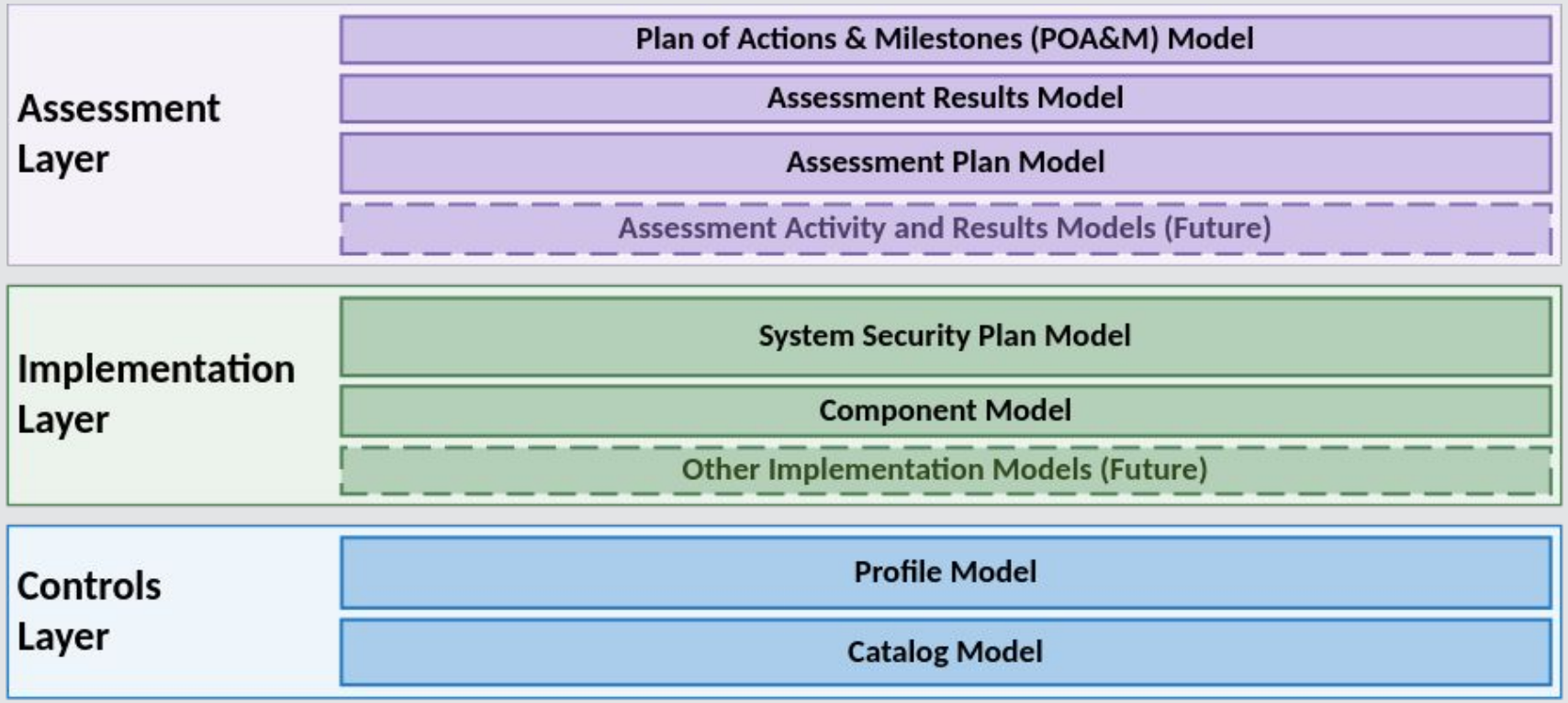
Declarative configuration

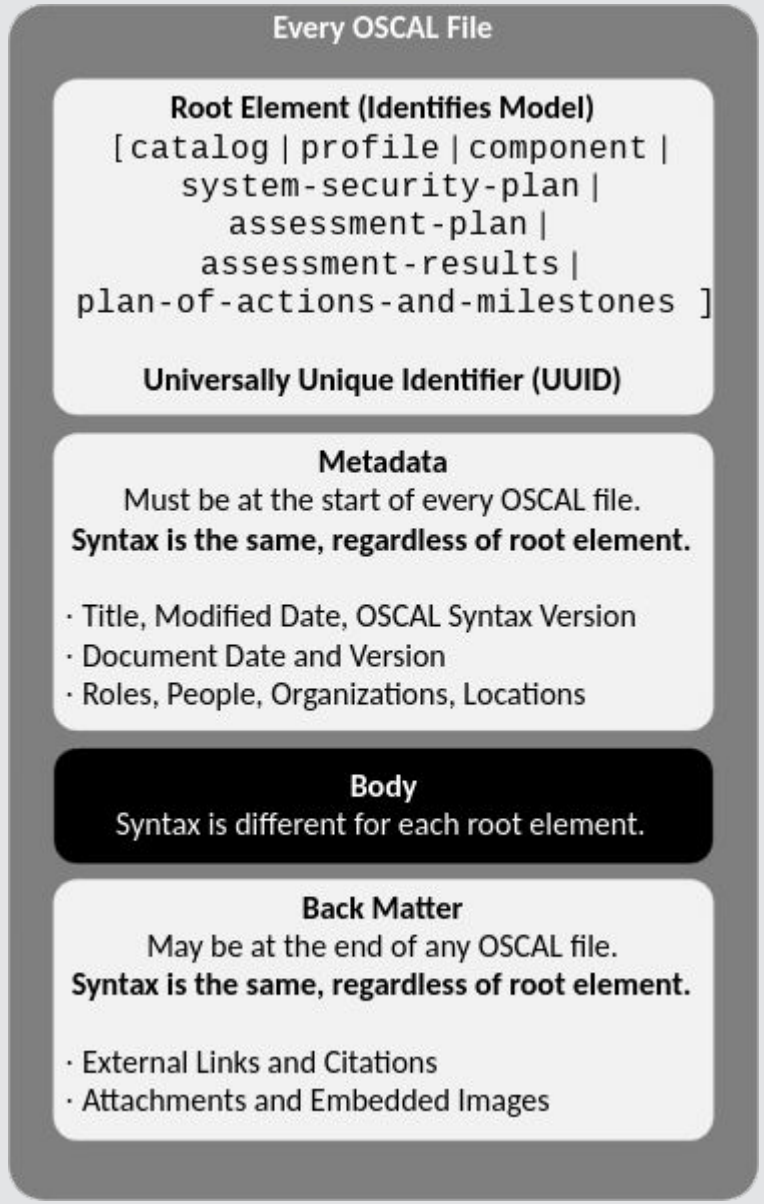
Kubernetes supports declarative control by specifying users' desired intent. The intent is carried out by asynchronous control loops, which interact through the Kubernetes API. This declarative approach is critical to the system's self-healing, autonomic capabilities, and application updates. This approach is in contrast to manual imperative operations or flowchart-like orchestration. [0]

Gitops vs. GRC?

[0] https://github.com/kubernetes/design-proposals-archive/blob/main/architecture/declarative-application-management.md#declarative-configuration

Why OSCAL?


Assessment Layer
Plan of Actions & Milestones (POA&M) Model
Assessment Results Model
Assessment Plan Model
Assessment Activity and Results Models (Future)
Implementation Layer
System Security Plan Model
Component Model
Other Implementation Models (Future)
Controls Layer
Profile Model
Catalog Model


Every OSCAL File
Root Element (Identifies Model)
[catalog | profile | component | system-security-plan | assessment-plan | assessment-results | plan-of-actions-and-milestones]
Universally Unique Identifier (UUID)
Metadata
Must be at the start of every OSCAL file.
Syntax is the same, regardless of root element.
· Title, Modified Date, OSCAL Syntax Version
· Document Date and Version
· Roles, People, Organizations, Locations
Body
Syntax is different for each root element.
Back Matter
May be at the end of any OSCAL file.
Syntax is the same, regardless of root element.
· External Links and Citations
· Attachments and Embedded Images

Kubernetes Policy + OSCAL = Compliance-as-Code

- PolicyReport CRD → OSCAL Assessment Result
  - RedHat and IBM contributions
- Platform One
  - All using and/or collaborating on OSCAL and using Kubernetes
  - Many examples in gitlab repos
- Defense Unicorns
  - Lula - Kyvero policy generation
- SLEDGEHammer: NIST 800-53 OSCAL + automation

Why do I need Compliance-as-Code?

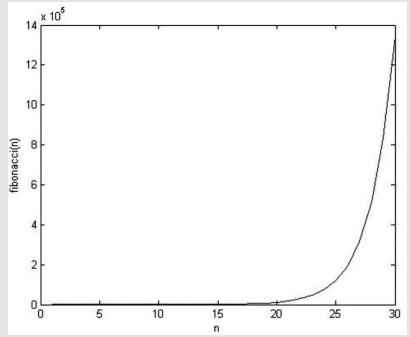
- Reactive too expensive: not enough time, people.
- Amplification of problems
  - one configuration mistake => amplified across templates
  - Amplified across deployments => 1000s of vulns
- Assessment and audits take months
- GRC vs. gitops (controls->components->SAR)
- Quick IRL example:
  - 19.7x cost to fix a problem in production vs PR-time
  - Recovery time - unknown; response time - unknown
  - Release frequency: 7 working days min
  - PR to deploy - huge variance 1-30+ days

Consistent, Codified Guard Rails

- User and Workload Identity
- Task and Workload Based Access + Networking: isolation
- Continuous Authorization
- Auditing
- Workload Inventory, Classification and Drift Detection
- Data Exposure Risk Inventory, Classification, and Leakage Detection
- Consistency across clusters - reduce cognitive load
- Aggregation for analytics Analytics and Benchmarking
- *FUTURE STATE*: Formal modeling
- *FUTURE STATE*: Predictive ML

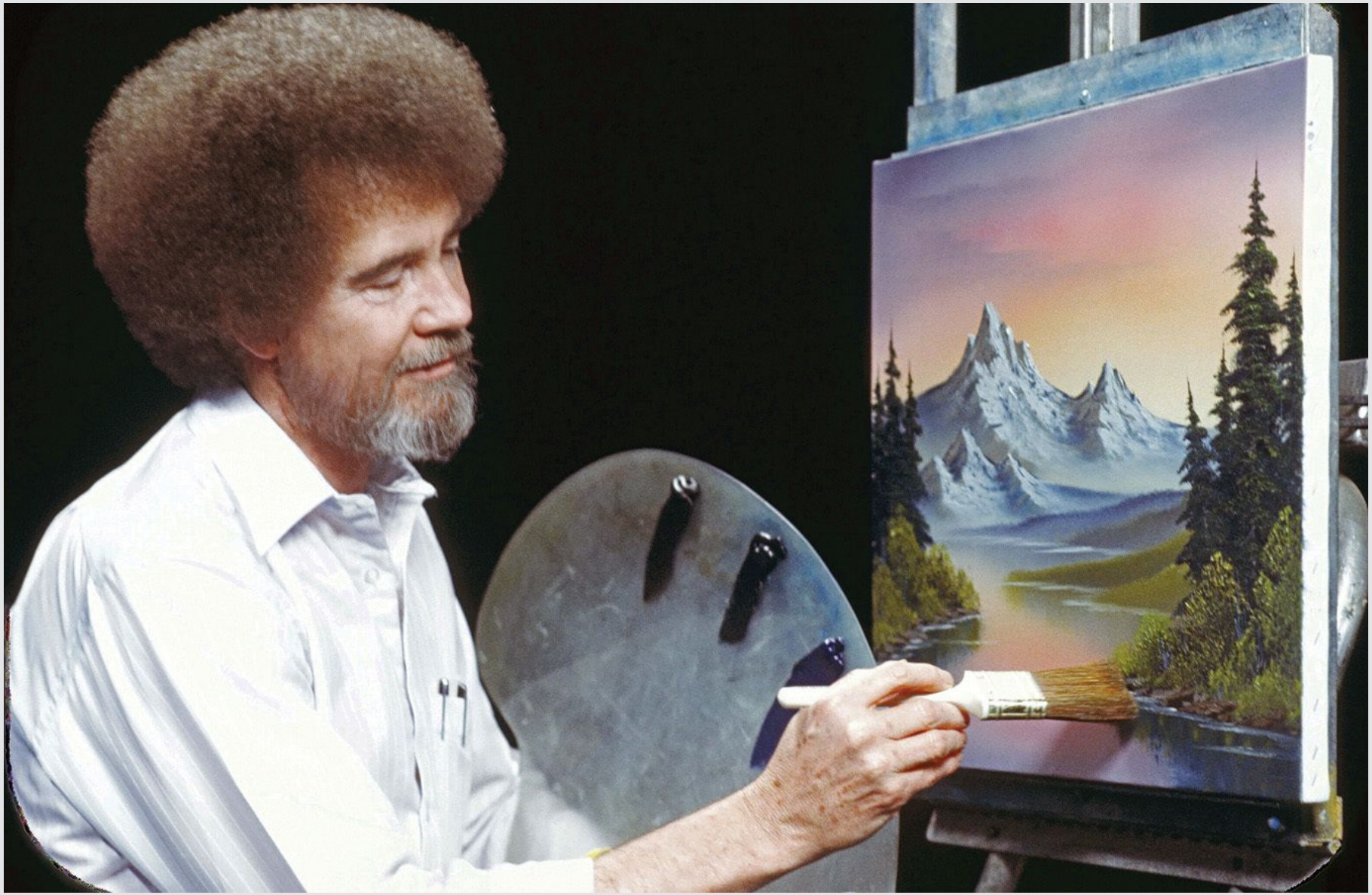
IRL Challenges

- Kubernetes great for loosely coupled apps
  - Many apps are monoliths - and that's fine for many use cases
- Kubernetes itself is evolving and best practices still being pressure tested
  - Service meshes and FaaS even more so
- Need "Policy Patterns"?
- "Fibonacci" ops…as # clusters+apps grow:
  - Complexity growth → exponential
  - Policy as a "spira mirabilis"

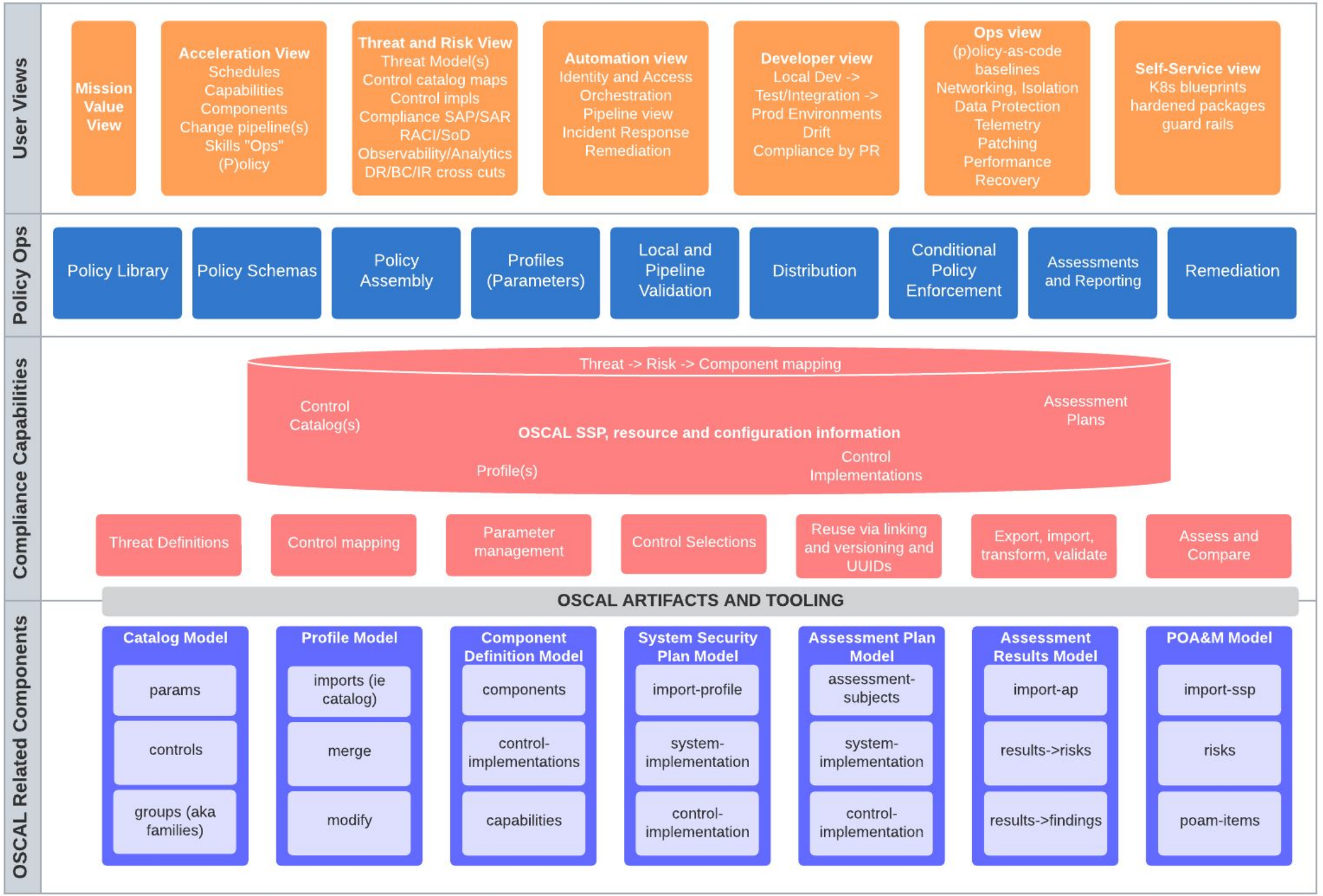


Utility

How useful is policy for X during Y?	Construction	Deployment Baseline	Normal Operations	SRE / Outliers	SecOps	Incident Response
Delivery Time	---	--	+	+	+++	+++
Separation of Concerns	+++	++	+++	++	+	+
Risk Reduction	+++	+++	+	+	+	+
Efficiency	---	-	-	---	+	+/-
Functionality	--	--	+/-	+	++	+++
Compliance	+++	+++	+++	--	++	+/-



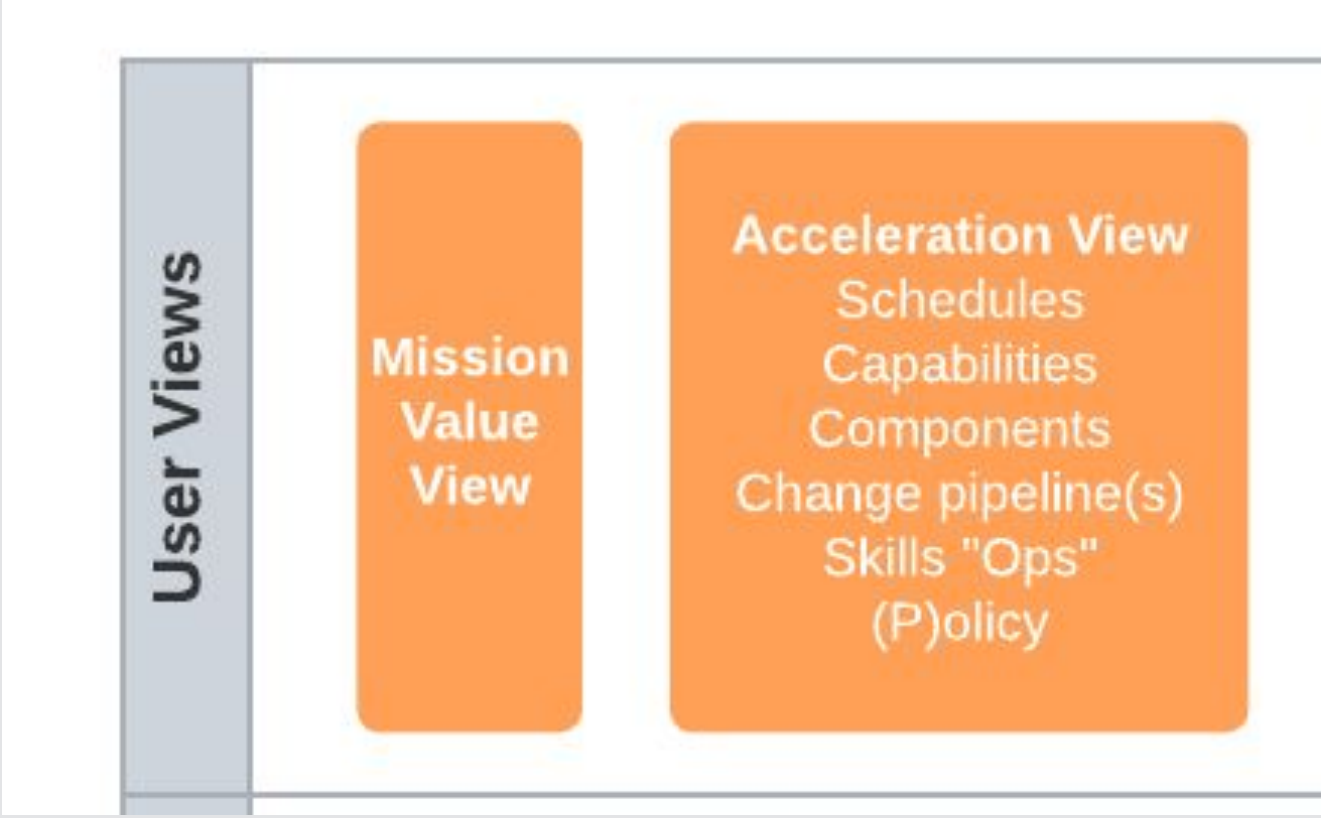
COMPLIANCE-AS-CODE CANVAS



Compliance Journey

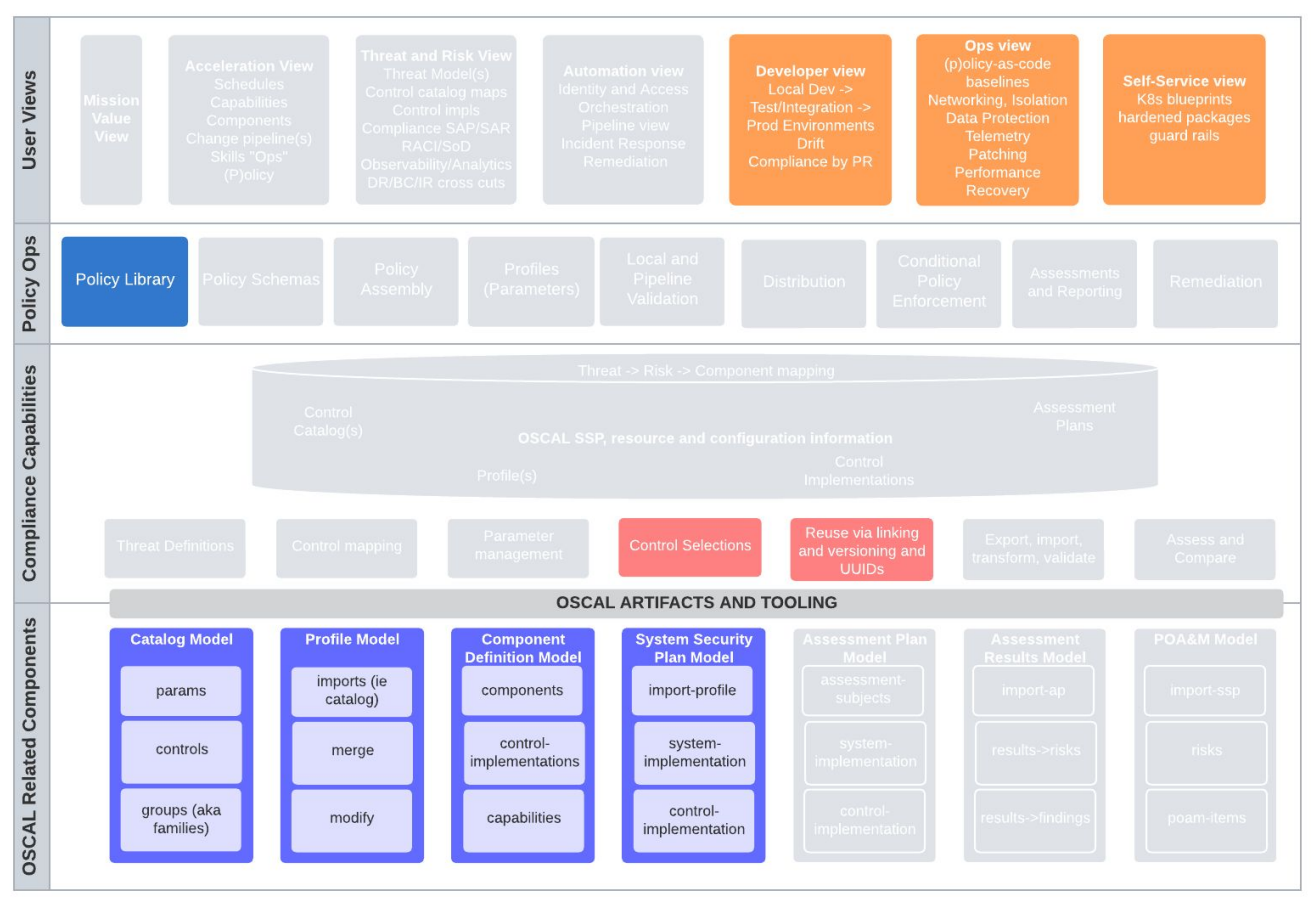
Sound familiar?

- Takes too long, too expensive to deliver new things…
- Kubernetes pilot project..
- Early adopter teams pile on, everyone makes things up as they go …
- Uh oh…something breaks (or gets hacked)... no one knows how to fix!!!
- Finger pointing
- Compliance too complex!



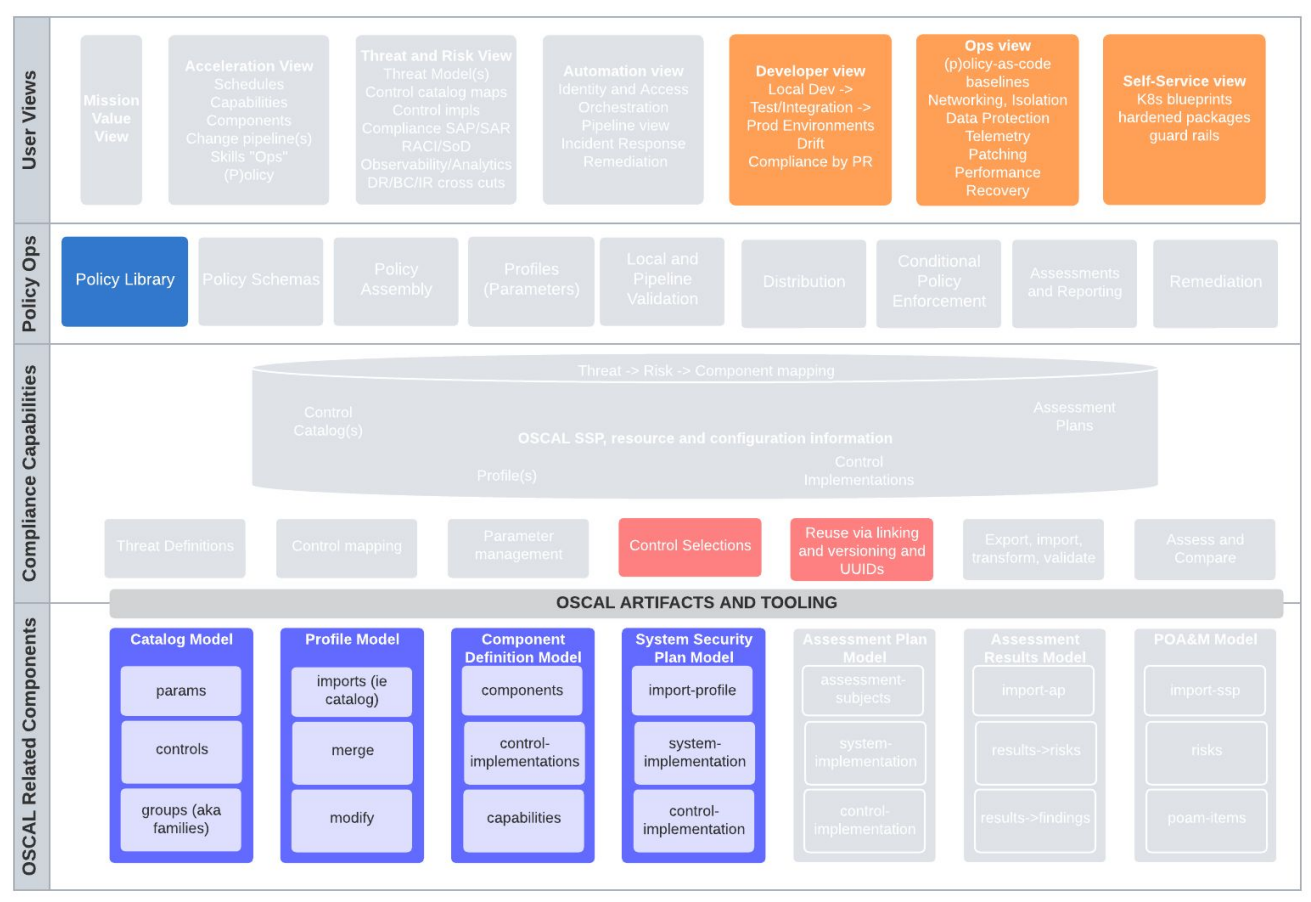
Policy Library

- Control Catalog and Profile
- Benchmarks and Baselines useful starting point
  - CIS Benchmarks
  - Preventive Controls - what APIs and how they are used, IaC constraints
  - Detective Controls - security features, alerts
- Organization around blueprints
  - Manage in git repo with PRs and branch management strategy



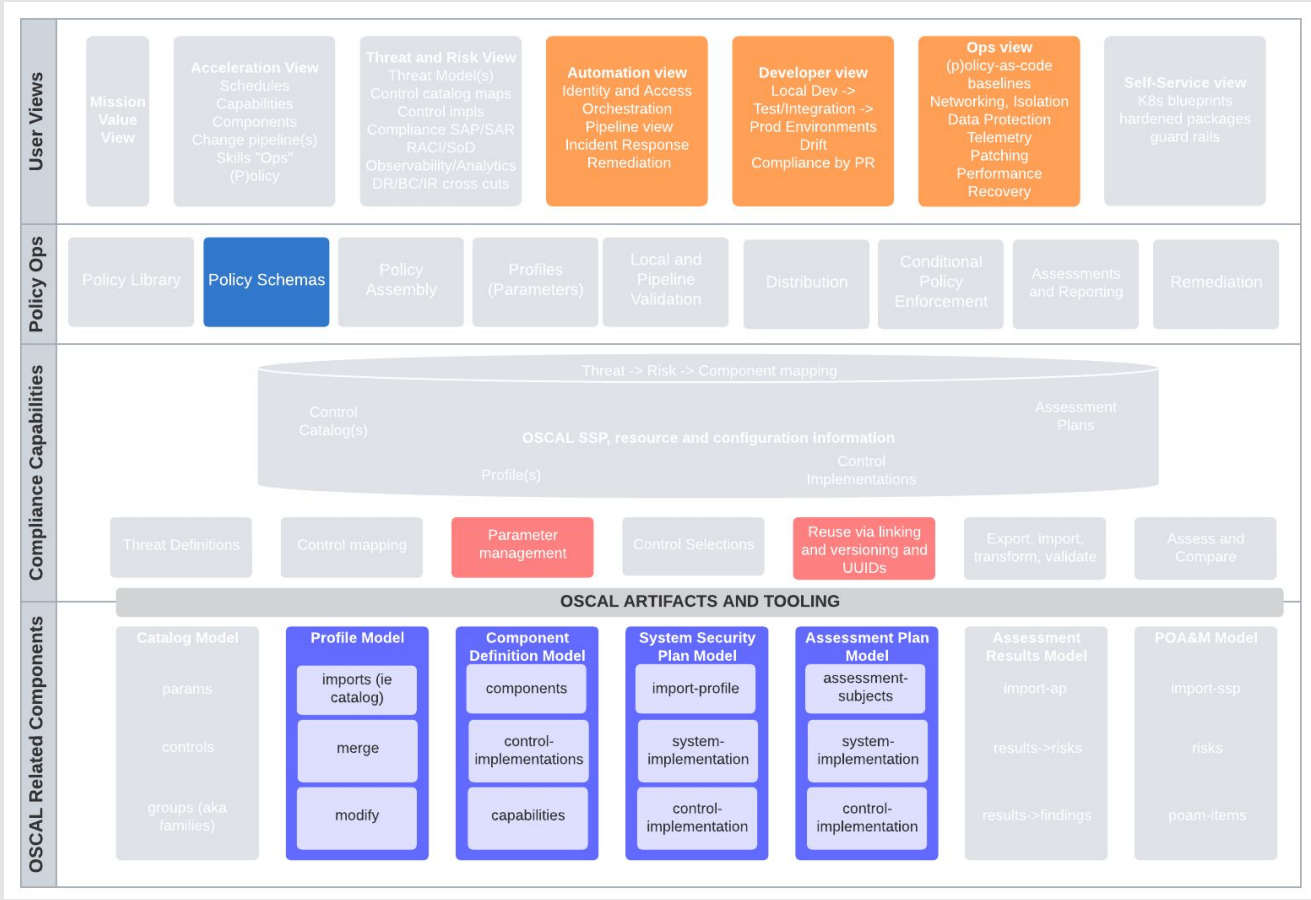
Kill and Yank

- Fork - initially simple, harder over time
- Patch - good for low frequency
- Composition
  - Pull - use APIs for distribution
  - Push policies to selected resources e.g. namespace and/or label selector
- Transformation - OSCAL itself does this
- Generation/DSL - Rego
- Automation: dynamic, aids encapsulation, mitigate ordering problems
- Parameterization: Gatekeeper



Policy Schemas

- Example: Gatekeeper ConstraintTemplate
- Kyverno rules written in YAML OpenAPIv3 schema --test
- OCM policy-templates
- CEL templates
- Blueprints provide connection to policy schemas
- Define unit tests for the policy templates
- Validate Policy Libraries using tests and pre-prod replay modeling

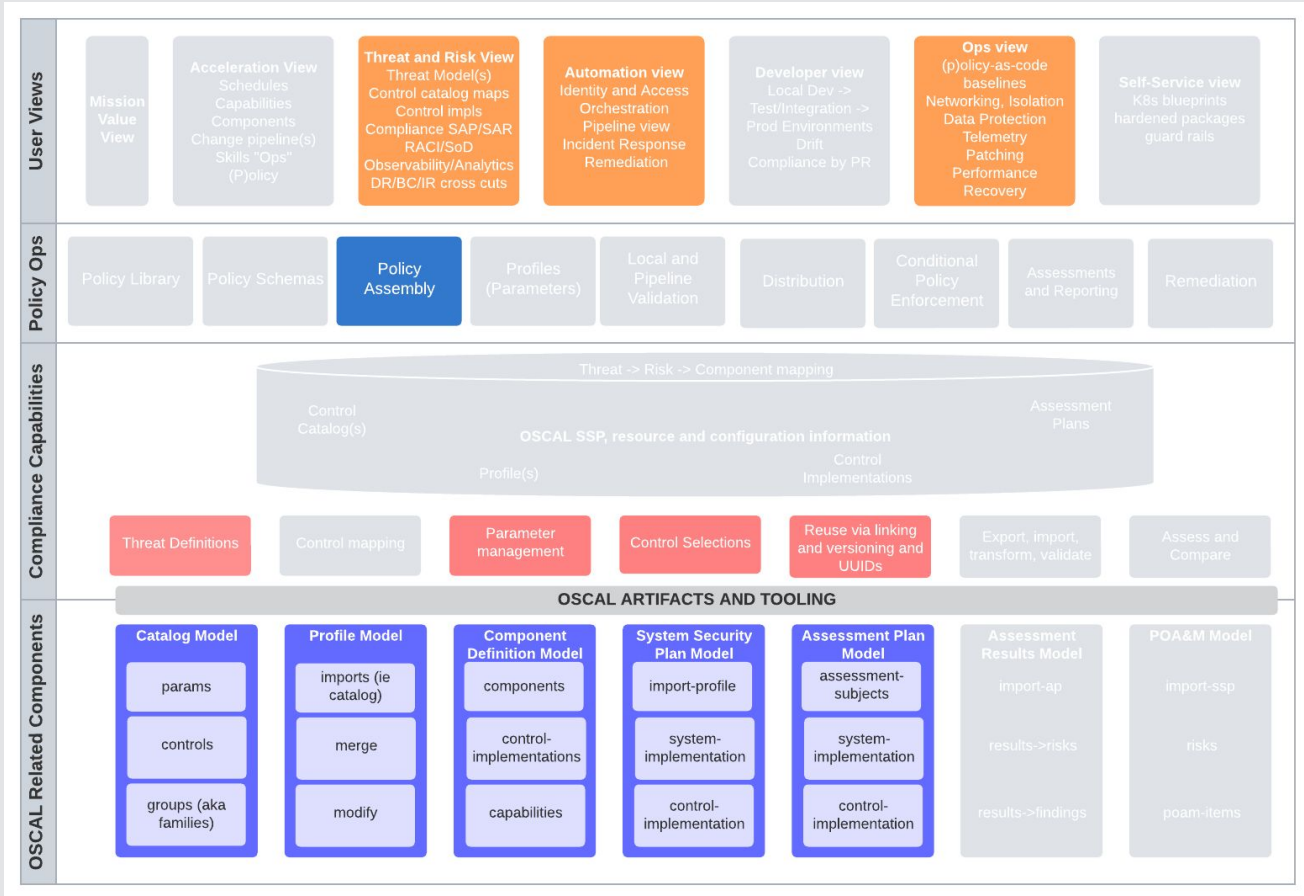


Policy Assembly

- DSL gen: control<->subject mapping
  - Ex: using TTPs to match to policy library (e.g. [dynamic policy composition])

```
applicable_policy := {
        "vm": "compute",
        "lambda": "compute",
        "container": "compute",
        "ip": "network",
        "securitygroup": "network",
        "waf": "network",
        "ssd": "storage",
        "volume": "storage" }
```

- Automation:
  - keyword regexes and heat map analysis
  - Using graphs e.g. K8s NetworkPolicies
- API based, eg. OpenAPI to Rego generation
- Hybrid - component “legos”
  - DSL+API “Functional” components:
    - control-implementations
    - assessment-subjects
  - Graph “security capabilities” - groups of control implementations in Component Model

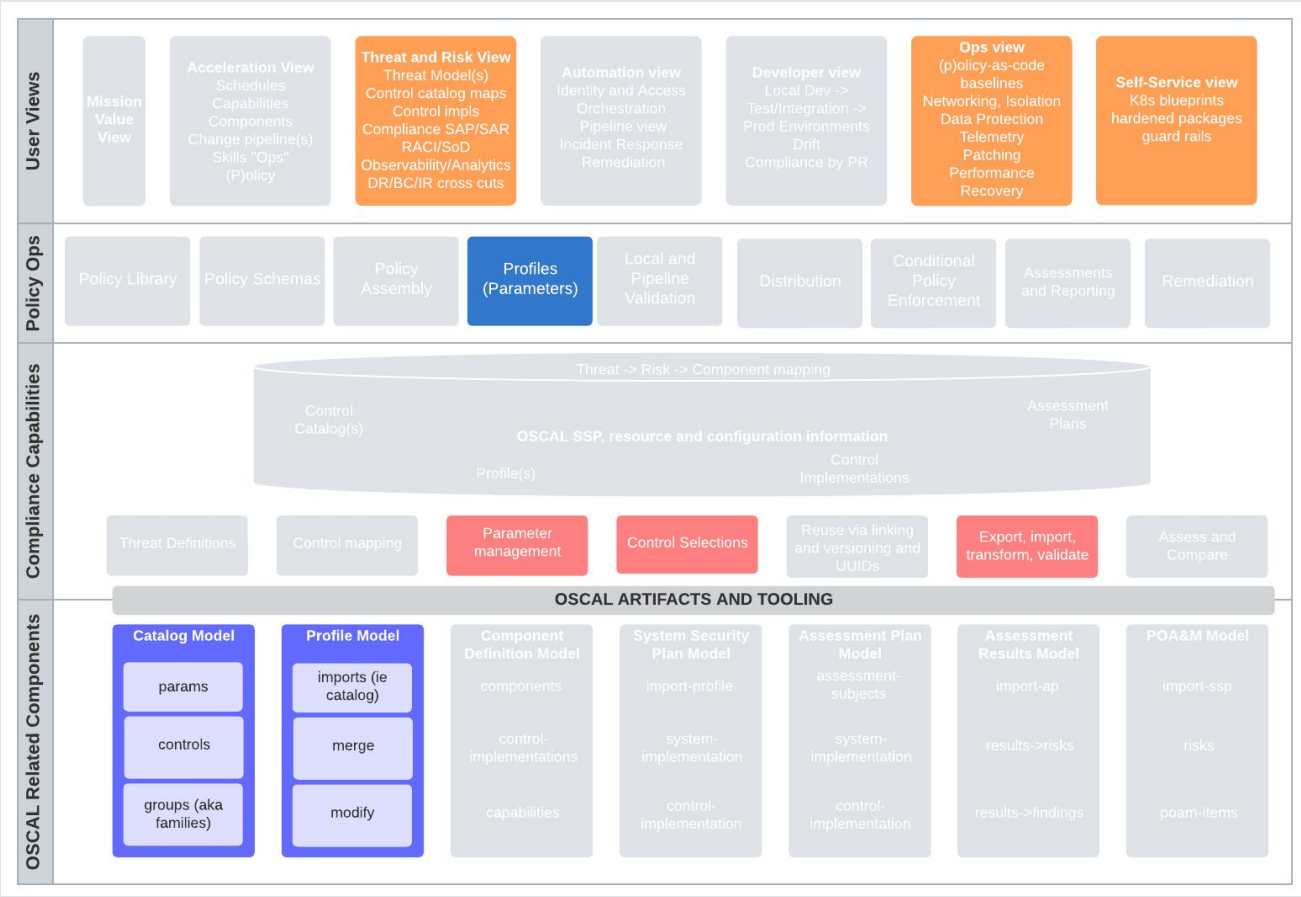


Profiles and Parameters

- Specific baseline controls + params
- Assemble parameterized policy

```
parameters:
  # repos <array>: The list of prefixes a container image is allowed to have.
  repos:
    - <string>
```

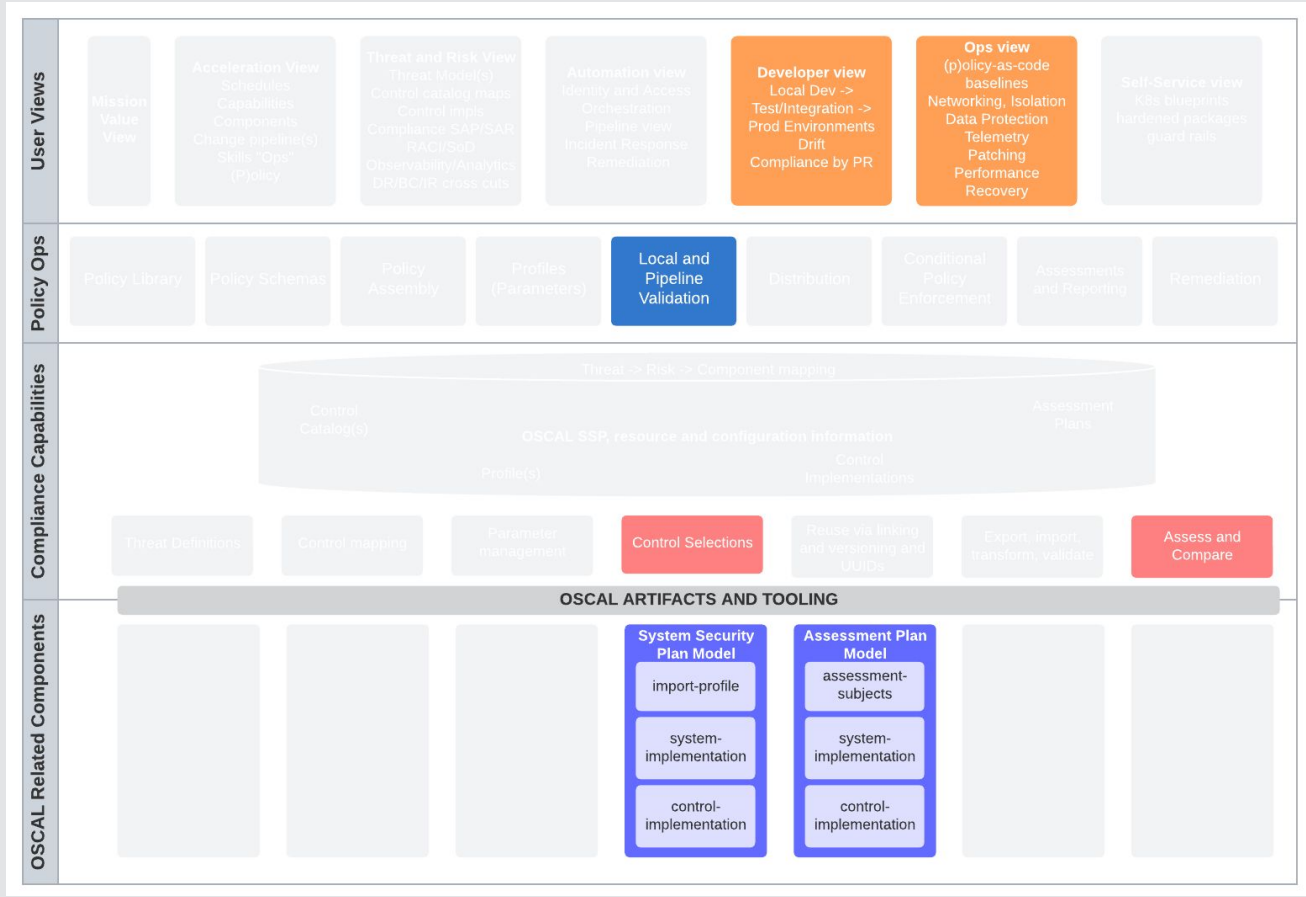
- OPA parameters, Kyverno Variables, CEL environment `cel.Variable`
- Ex: Mutating webhook AC for adding values to constraints



Here Be Dragons: Parameterization easy at first, difficult to maintain over time.

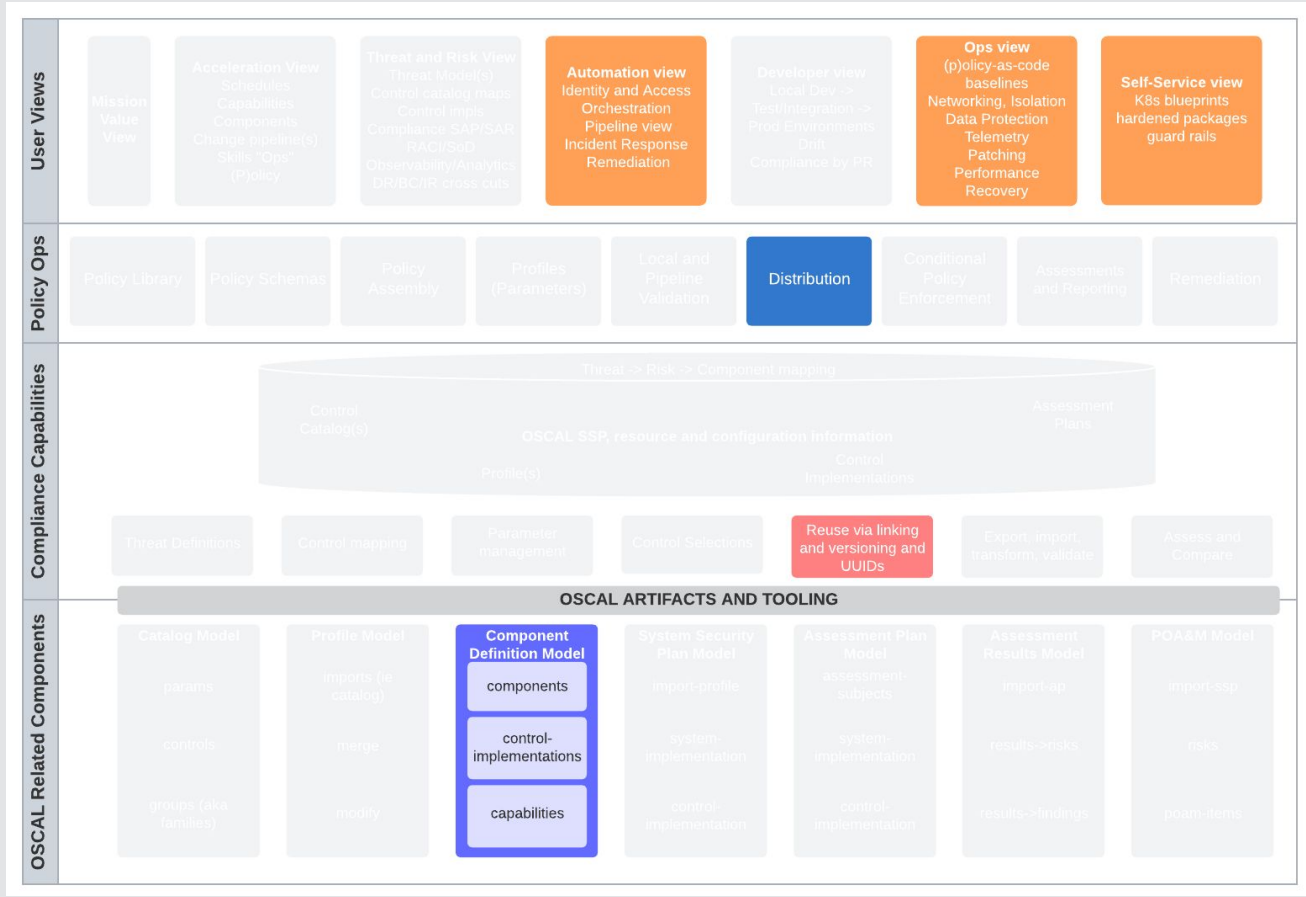
Policy Validation

- Local Tests
- Unit Tests
- Pipeline testing
- Mocking
- Pre-Prod Tests against cluster
- Replay Testing from Prod logs
- Coverage



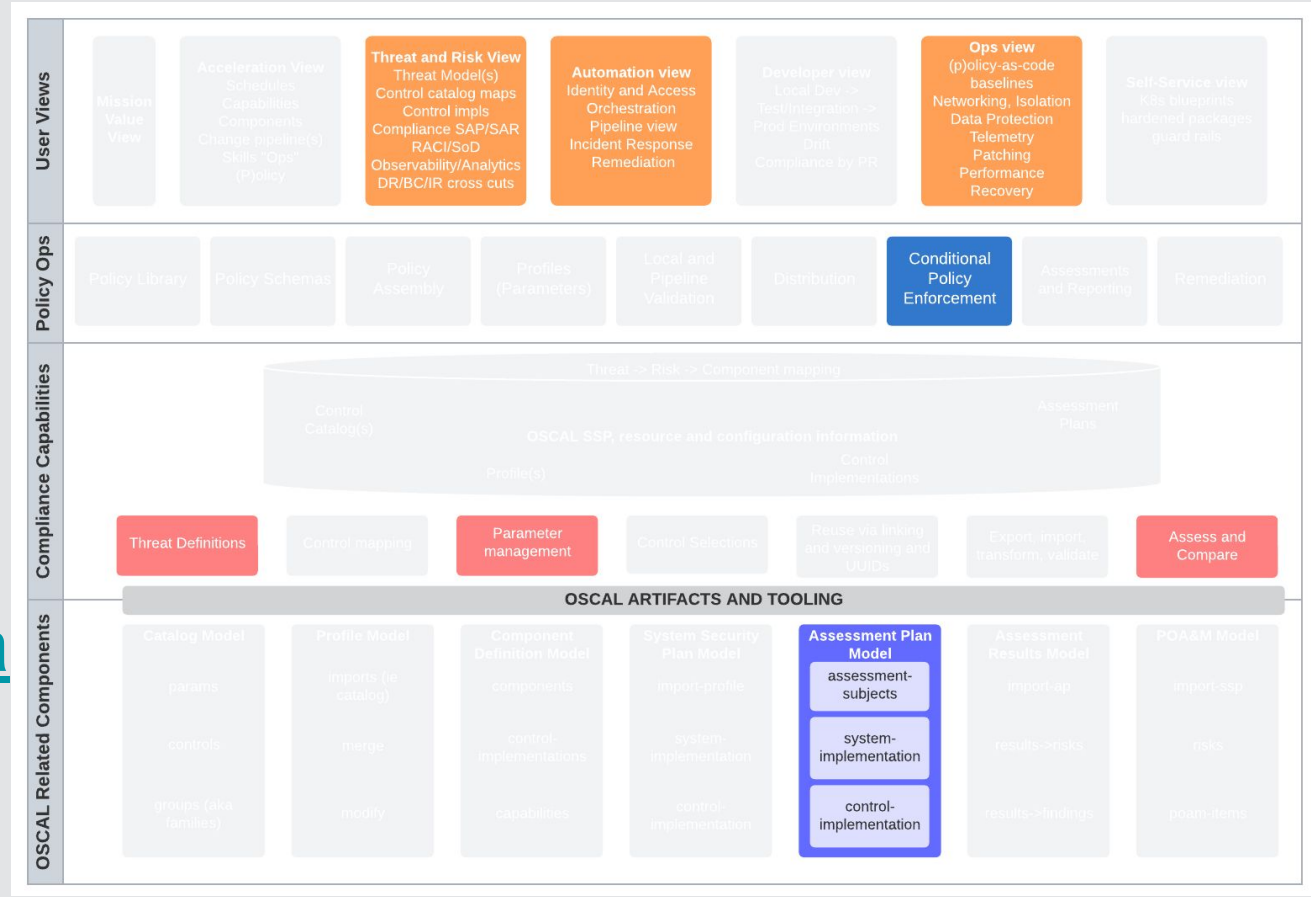
Policy Distribution

- Policy Bundle Registry (e.g. OPA API, cloud storage)
- Git repo
- Helm chart repositories
- OPA can consume policy bundles packaged as OCI images
- OCM Placement
- Cloud specific (GCP Config, AWS S3/SSM, Azure Policy extension)



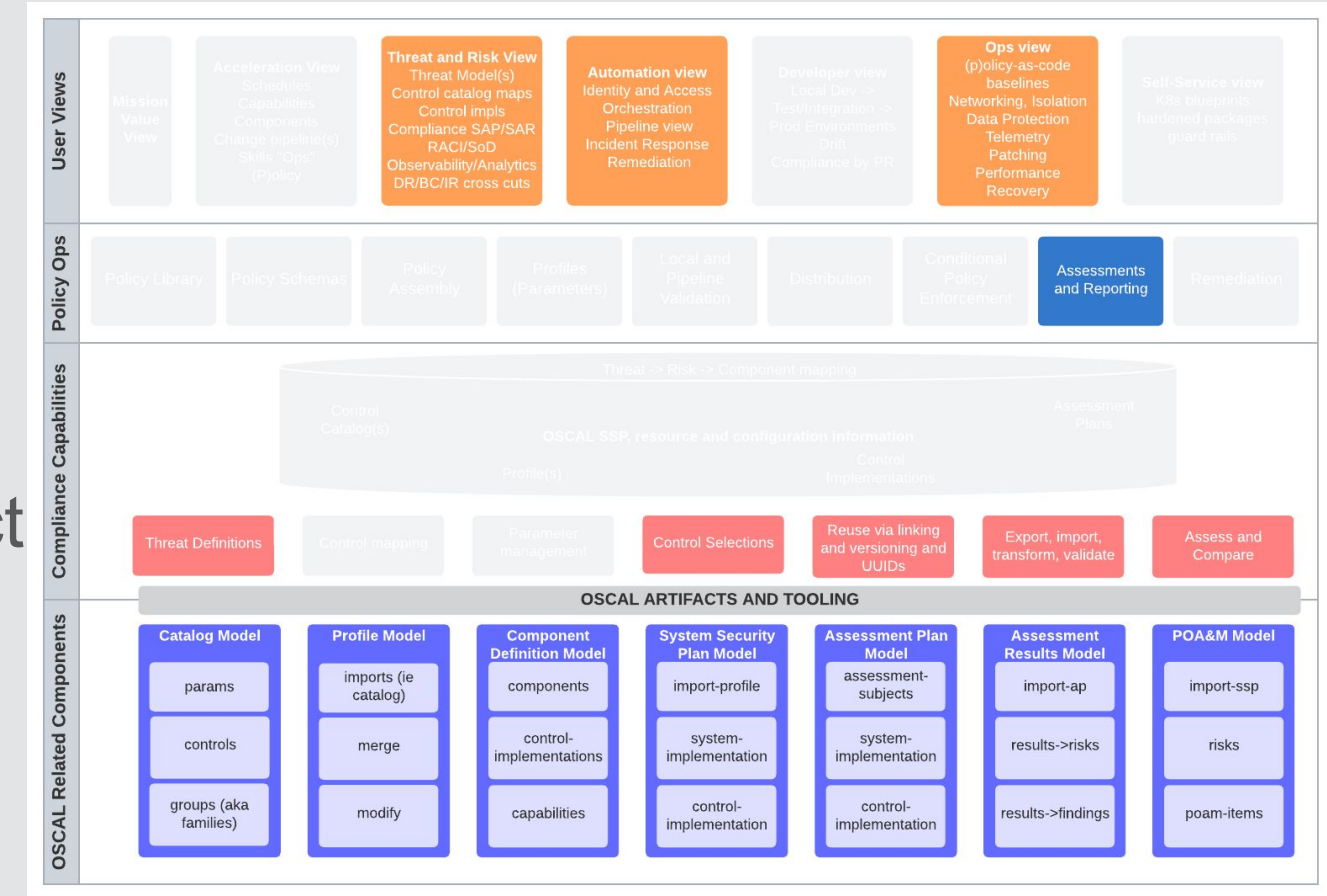
Conditional Policy Enforcement

- Using External Data during policy decision eval
- How:
  - OPA external data pull, send
  - Gatekeeper external data Providers
  - Kyverno OCI image metadata
  - OCM Secrets
- Examples: verifying container signatures, per-tenant policies



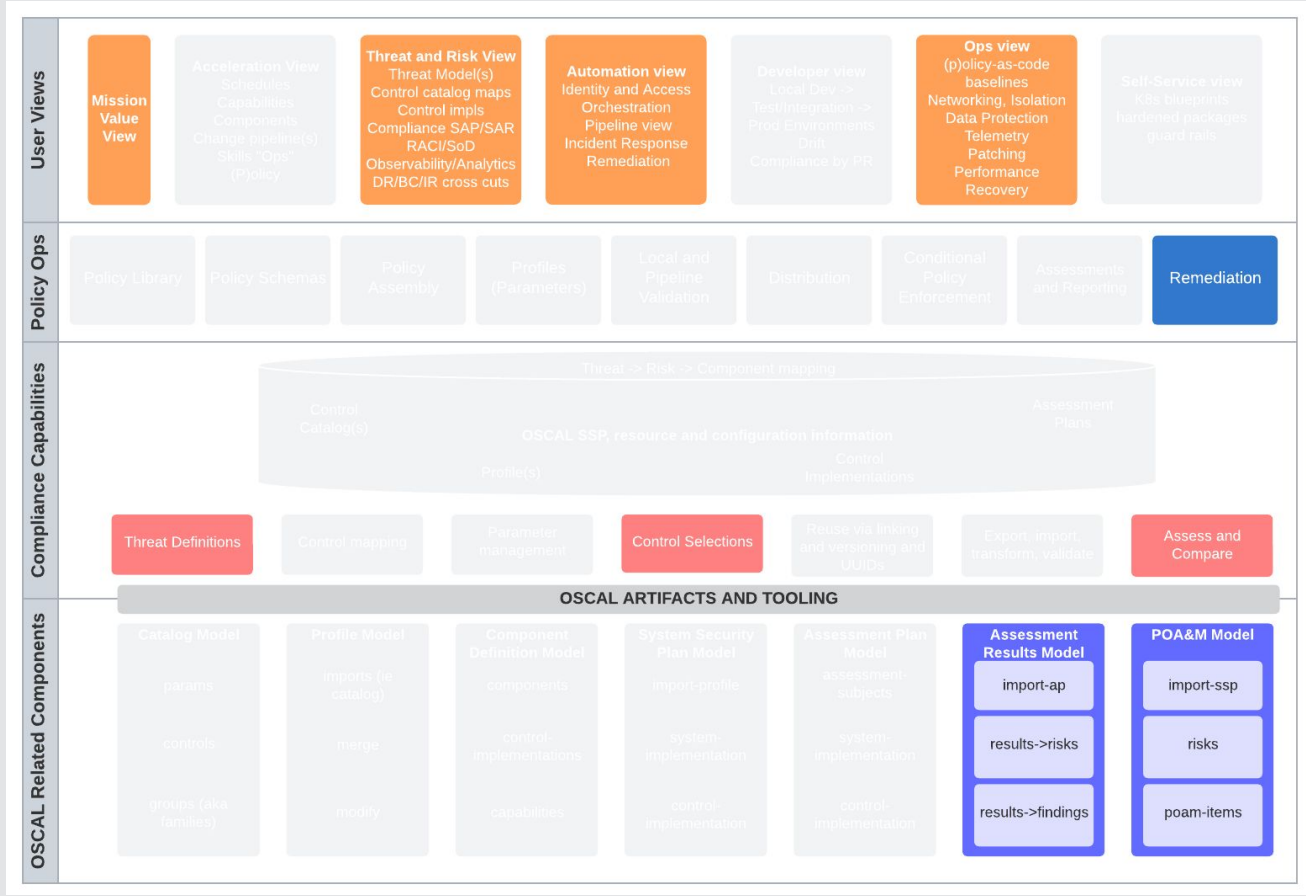
Policy Assessment and Reporting

- PolicyReport CRD - e.g. Kyverno and OCM - transform to SAR
- OPA/Gatekeeper Audit, Prometheus Metrics
- Do you need a GUI or git?
  - SAR is another OSCAL artifact
  - can be managed in git or cloud BLOB store



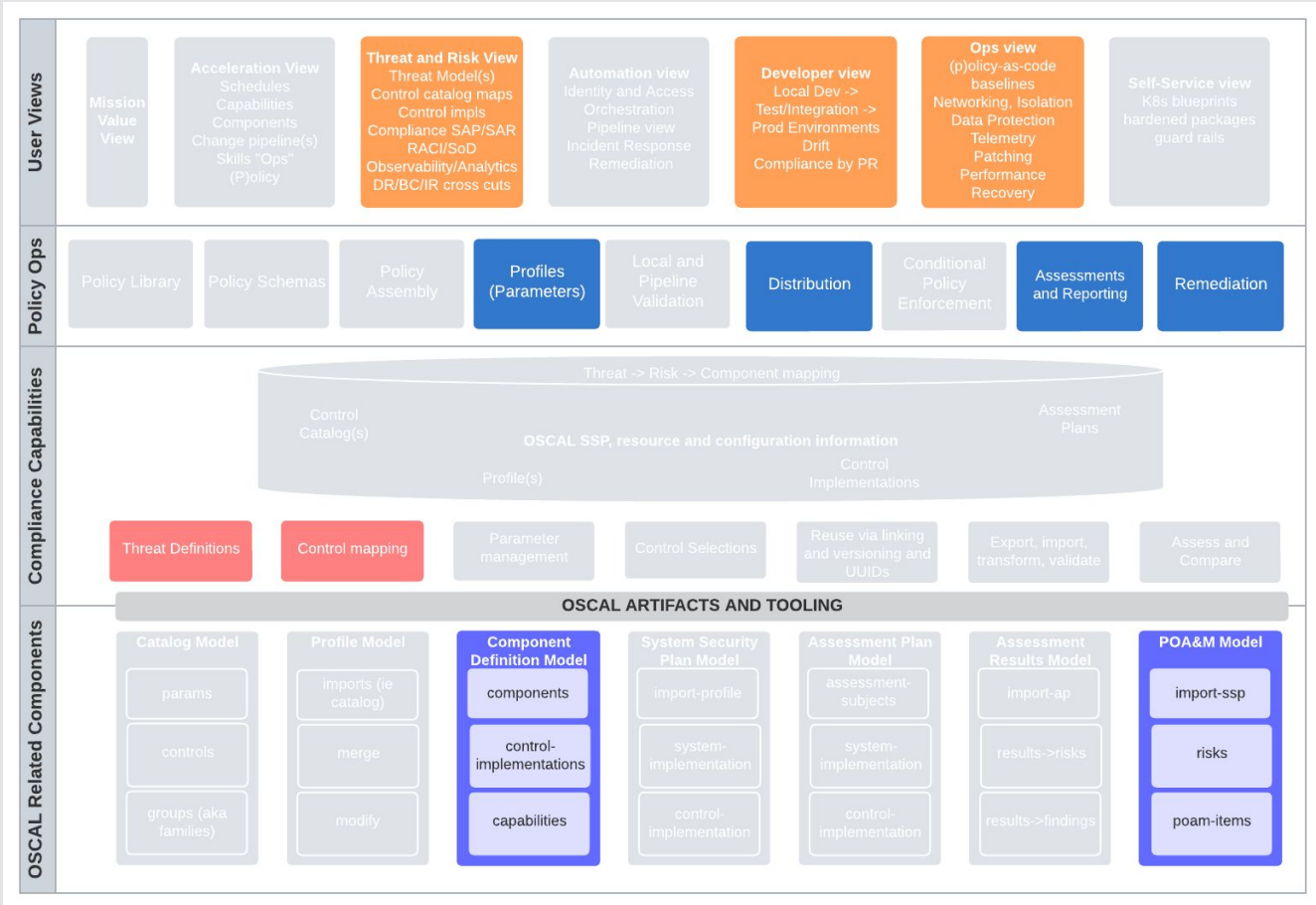
Automated Remediation (Policy Adjacent)

- Remediate Drift with Controllers or sidecars, eg. CloudCustodian
- PolicyReport -> PR gen:
  - Sandbox/Isolate, Logging/Telemetry++
- Mutate Labels for TTPs
- Generate POA&M Risk Log and Open Risks



Cross Cutting Concerns

- Heuristics, eg govCAR
- Threat Model, eg MITRE ATT&CK
- Static Analysis
- Formal Methods
- Graph ML, LLMs



What's Next?

- More IRL examples
- Operators using
- 3rd party audits
- More better tools

- Community involvement!
  - SLEDGEHammer

Please scan the QR Code above
to leave feedback on this session