DGOV DTT Architecture Decision Records

DGOV DTT

Contents

DGOV DTT Architecture Decision Records 2

Architecture Principles 3

Security ADRs 3

ADR 001: Application Isolation 3

ADR 005: Secrets Management 4

ADR 008: Email Authentication Protocols 5

ADR 011: AI Tool Governance 6

ADR 012: Privileged Remote Access 7

ADR 013: Identity Federation Standards 9

ADR 016: Web Application Edge Protection 12

Operations ADRs 13

ADR 002: AWS EKS for Cloud Workloads 13

ADR 006: Automated Policy Enforcement 14

ADR 007: Centralised Security Logging 15

ADR 010: Infrastructure as Code 16

ADR 014: Object Storage Backups 18

ADR 015: Data Governance Standards 19

ADR 017: Analytics Tooling Standards 20

ADR 018: Database Patterns 20

Development ADRs 21

ADR 003: API Documentation Standards 21

ADR 004: CI/CD Quality Assurance 22

ADR 009: Release Documentation Standards 23

Reference Architectures 25

Reference Architecture: Content Management 25
Reference Architecture: Data Pipelines 27
Reference Architecture: Identity Management 28
Reference Architecture: OpenAPI Backend 31

Templates 32
ADR ###: Specific Decision Title 32
Reference Architecture: Pattern Name 33
Contributing Guide 34
Glossary 38

DGOV DTT Architecture Decision Records

Architecture records and current decision state to support infrastructure and platform operations for Office of Digital Government (DGOV) Digital Transformation and Technology Unit (DTT).

Supporting training material is available at the DGOV Technical - DevSecOps Induction (guided by the WA Cyber Security Policy).

The Architecture Principles guide all decisions documented here.

Structure

This project uses mdBook to generate documentation from markdown files:

- **ADRs** are organized in directories by domain (`development/`, `operations/`, `security/`)
- **Navigation** is defined in `SUMMARY.md`
- **Build output** goes to `book/` directory

Quick Start

```
just setup     # One-time setup
just serve     # Preview locally (port 8080)
```

Run `just` to see all available commands.

View as PDF | **Browse online**

Development

First time setup requires installing tools - run `just setup` for automated installation.

Contributing

New ADRs follow a structured format documenting the **context** (problem), **decision** (solution), and **consequences** (risks).

1. Create new markdown file in appropriate directory (e.g., `security/019-new-topic.md`)
2. Add entry to `SUMMARY.md` in the correct section
3. Follow the ADR template structure

See the Contributing Guide for complete workflow and quality standards.

Architecture Principles

Status: Accepted | **Date:** 2025-03-07

1. Establish secure foundations

Integrate security practices **from the outset**, and throughout the design, development and deployment of products and services, per the ACSC Foundations for modern defensible architecture.

2. Understand and govern data

Use authoritative data sources to ensure data consistency, integrity, and quality. Embed data management and governance practices, including information classification, records management, and Privacy and Responsible Information Sharing, throughout information lifecycles.

3. Prioritise user experience

Apply user-centered design principles to simplify tasks and establish intuitive mappings between user intentions and system responses. Involve users throughout design and development to iteratively evaluate and refine product goals and requirements.

4. Preference tried and tested approaches

Adopt sustainable opensource software, and mature managed services where capabilities closely match business needs. When necessary, bespoke service development should be **led by internal technical capabilities** to ensure appropriate risk ownership. Bespoke software should **preference open standards and code** to avoid vendor lock-in.

5. Embrace change, release early, release often

Design services as **loosely coupled** modules with **clear boundaries and responsibilities**. Release often with tight feedback loops to test assumptions, learn, and iterate. Enable frequent and predictable high-impact changes (your service does not deliver or add value until it is in the hands of users) per the CNCF Cloud Native Definition.

6. Default to open

Encourage transparency, inclusivity, adaptability, collaboration, and community by defaulting to permissive licensing of code and artifacts developed with public funding.

Security ADRs

ADR 001: Application Isolation

Status: Accepted | **Date:** 2025-02-17

Context

Not isolating applications and environments can lead to significant security risks. The risk of lateral movement means threats of vulnerability exposure of a single application can compromise other applications or the entire environment. This lack of isolation can enable the spread of malware, unauthorised access, and data breaches.

- Open Web Application Security Project (OWASP) Application Security Verification Standard (ASVS)
- Australian Cyber Security Centre (ACSC) Guidelines for System Hardening

Decision

To mitigate the risks associated with shared environments, all applications and environments should isolate by default. This isolation can achieve through the following approaches:

1. **Dedicated Accounts**: Use separate cloud accounts / resource groups for different environments (for example, development, testing, production) to ensure complete isolation of resources and data.
2. **Kubernetes Clusters**: Deploy separate Kubernetes clusters for different applications or environments to isolate workloads and manage resources independently.
3. **Kubernetes Namespaces**: Within a Kubernetes cluster, use namespaces to logically separate different applications or environments, providing a level of isolation for network traffic, resource quotas, and access controls.

The preferred approach for isolation should drive by data sensitivity and product boundaries.

Consequences

If applications and environments are not isolated by default, the following consequences may arise:

1. **Increased Risk of Compromise**: A vulnerability in one application can lead to the compromise of other applications or the entire environment.
2. **Difficulty in Incident Response**: Without isolation, it becomes challenging to contain and mitigate security incidents.
3. **Compliance Issues**: Failure to isolate environments may lead to non-compliance with regulatory requirements and industry standards.
4. **Data Breaches**: Sensitive data may expose or stolen due to unauthorised access from the lack of isolation.

By adopting this decision, we aim to enhance the security posture of our systems, reduce the risk of security incidents, and ensure compliance with relevant standards and regulations.

ADR 005: Secrets Management

Status: Accepted | **Date:** 2025-02-25

Context

Per the Open Web Application Security Project (OWASP) Secrets Management Cheat Sheet:

> Organizations face a growing need to centralize the storage, provisioning, auditing, rotation and management of secrets to control access to secrets and prevent them from leaking and compromising the organization. Often, services share the same secrets, which makes identifying the source of compromise or leak challenging.

To address these challenges, we need a standardised, auditable approach to managing and rotating secrets within our environments. Secrets should be accessed at runtime by workloads and should never be hard-coded or stored in plain text.

- Amazon Web Services (AWS) Secrets Manager Compliance validation
- Using Elastic Kubernetes Service (EKS) encryption provider support for defense-in-depth

Decision

Use AWS Secrets Manager to store and manage secrets.

- Secrets should be fetched and securely injected into AWS resources at runtime.
- The secret rotation period (lifetime) must be captured in the system design.
  - Rotate secrets automatically where possible, or ensure that a manual rotation process is documented and followed.
- Use Identity and Access Management (IAM) policy statements to use least-privilege access to secrets.
- ADR 002: AWS EKS for Cloud Workloads kubernetes workloads should use EKS Key Management Service (KMS) secrets encryption with namespace local secrets by default.
  - If secrets need to be accessed by several clusters, use External Secrets Operator to synchronise them from the primary secret store in AWS Secrets Manager.

Consequences

Positive:

- **Automated Management**: Reduces human error and ensures consistent updates.
- **Compliance**: Meets auditing and compliance requirements.

Negative:

- **Dependency on AWS**: Using AWS Secrets Manager for all secrets could make future migrations of AWS difficult. Ensuring secret rotation is straightforward and documented should minimise this consequence.

Risks of not implementing:

- **Security Risk**: Regular handling or manual handling of secrets increases exposure risk.
- **Operational Overhead**: Manual processes for configuring and rotating secrets can be error-prone and inefficient.

By implementing this decision, we aim to enhance the security and efficiency of our secret management processes, ensuring that sensitive information is handled securely and automatically.

ADR 008: Email Authentication Protocols

Status: Proposed | **Date:** 2025-07-22

Context

Government email domains are prime targets for cybercriminals who exploit them for phishing attacks, business email compromise, and brand impersonation. Citizens and businesses expect government emails to be trustworthy, making email authentication critical for maintaining public confidence and preventing fraud.

Without proper email authentication, attackers can easily spoof government domains to conduct social engineering attacks, distribute malware, or harvest credentials from unsuspecting recipients.

References:

- ACSC How to combat fake emails

Decision

Implement email authentication standards for all government domains:

Required Standards:

- **SPF**: Publish records defining authorized mail servers with strict policies (“~all” or “-all”)
- **DKIM**: Sign all outbound email with minimum 2048-bit RSA keys, rotate annually

- **DMARC**: Progress from “p=none” to “p=reject” with subdomain policies and reporting
- **BIMI**: Implement verified brand logos with Verified Mark Certificates (VMCs)

Implementation:

- Monitor DNS records for tampering
- Regular authentication testing and effectiveness reviews
- Incident response procedures for authentication failures
- Integration with email security gateways

Consequences

Risks of not implementing:

- Email spoofing and phishing attacks using government domains
- Brand reputation damage and reduced email deliverability
- Compliance violations with security requirements

Benefits:

- Significant reduction in email-based attacks
- Enhanced brand protection and email trust
- Improved compliance and threat visibility

ADR 011: AI Tool Governance

Status: Proposed | **Date:** 2025-07-29

Context

AI tools across all organisational functions process sensitive data and can make automated decisions affecting security, privacy, and compliance. Without governance ensuring human oversight of AI decisions, these tools pose significant risks including unauthorized data exposure, biased decision-making, and compliance violations. The highest risk areas are AI tools that process government data or make decisions without explicit human approval.

Current high-risk scenarios include:

- **Automated Decision-Making**: AI tools making policy, approval, or resource allocation decisions without human review
- **Government Data Processing**: Sensitive organisational data processed by offshore AI services
- **Uncontrolled AI Outputs**: AI-generated content, code, or analysis used without human validation
- **Privacy Violations**: Personal information processed by AI without appropriate consent or controls

References:

- ACSC Information Security Manual (ISM)
- WA Cyber Security Policy
- Privacy Act 1988

Decision

Implement mandatory human oversight for all AI tool usage with pre-approval for any AI tools that process organisational data or generate outputs used in official capacity.

Human Oversight Requirements:

All AI tools must ensure:

- **Human-in-the-Loop**: No automated AI decisions without explicit human review and approval
- **Output Validation**: All AI-generated content must be reviewed by qualified humans before use
- **Decision Accountability**: Clear human responsibility for all AI-assisted decisions
- **Override Capability**: Humans must be able to override or reject AI recommendations

Covered AI Tools:

- Development and coding assistance tools
- Content generation and writing assistants
- Data analysis and business intelligence platforms
- Decision support and recommendation systems
- Customer service and communication chatbots
- Document processing and workflow automation

Implementation Examples:

- **Approved**: Independently assured developer tools with mandatory code review processes for all AI-generated code
- **Rejected**: Automated HR screening tools that make hiring decisions without human review

Mandatory Isolation Requirements:

Any approved AI tools must be fully sandboxed with no access to:

- Production credentials or secrets
- Privileged system accounts
- Sensitive data repositories
- Network-privileged environments
- Administrative interfaces

Technical Implementation:

- AI tools must run in isolated environments with minimal permissions
- No network access to internal systems or databases
- Separate service accounts with restricted access scopes
- Regular audit of AI tool permissions and data access
- Clear data classification and handling procedures
- Automated detection of credential or sensitive data exposure

Consequences

Risks of not implementing:

- **Unaccountable Decisions**: AI making critical decisions without human oversight or review
- **Data Exposure**: Sensitive government data processed by offshore AI services
- **Compliance Violations**: Breach of Privacy Act and data sovereignty requirements
- **Operational Errors**: Unchecked AI outputs causing incorrect policy or technical decisions

Benefits of implementation:

- Human accountability for all AI-assisted decisions
- Compliance with privacy and data sovereignty requirements
- Reduced risk of AI-generated errors affecting operations
- Clear audit trail demonstrating responsible AI usage

ADR 012: Privileged Remote Access

Status: Proposed | **Date:** 2025-07-22

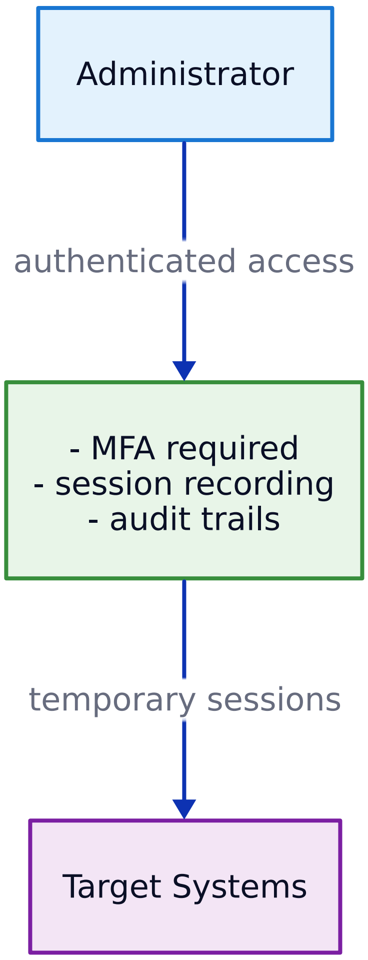
Context

Traditional privileged access methods using jump boxes, bastion hosts, and shared credentials create security risks through persistent network connections and broad administrative access. Modern cloud-native alternatives provide better security controls and audit capabilities for administrative tasks.

- ACSC Information Security Manual (ISM)
- WA SOC Cyber Network Management Guideline

Decision

Replace traditional bastion hosts and jump boxes with cloud-native privileged access solutions:



Prohibited Methods:

- Bastion hosts and jump boxes with persistent SSH access
- Direct SSH/RDP access to production systems
- Shared administrative credentials and keys

- VPN-based administrative access

Required Access Methods:

- **Server Access**: AWS Systems Manager Session Manager (replaces SSH to EC2)
- **Infrastructure Management**: AWS CLI with temporary credentials (replaces persistent VPN)
- **Kubernetes Access**: kubectl with IAM authentication (replaces cluster SSH)
- **Infrastructure Deployment**: Terraform Cloud with audit trails (replaces manual deployment)

Access Controls:

- Multi-factor authentication for all access
- Time-limited sessions (maximum 4 hours)
- Identity-based access through cloud IAM
- Approval workflows for privileged access
- Session recording and audit logging

Implementation:

- All sessions initiated through APIs only
- Short-lived credentials (maximum 1-hour validity)
- Real-time monitoring and alerting
- Integration with SIEM systems
- Emergency break-glass procedures with full audit

Consequences

Risks of not implementing:

- Unauthorized lateral movement through network connections
- Persistent access leading to prolonged breaches
- Non-compliance with zero-trust principles

Benefits:

- Significantly reduced attack surface
- Enhanced audit capabilities and compliance
- Better credential security through short-lived tokens

ADR 013: Identity Federation Standards

Status: Proposed | **Date:** 2025-07-29

Context

Applications need to integrate with multiple identity providers including jurisdiction citizen identity services, enterprise directories, and cloud identity platforms. Current approaches use inconsistent protocols (SAML, OIDC, proprietary) creating integration complexity and security inconsistencies.

Modern identity federation requires support for emerging standards like verifiable credentials while maintaining compatibility with legacy enterprise systems.

- Digital ID Act 2024
- OpenID Connect Core 1.0
- OWASP Authentication Cheat Sheet

Decision

Standardize on OpenID Connect (OIDC) as the primary federation protocol for all new identity integrations, with SAML 2.0 support only for legacy systems that cannot support OIDC.

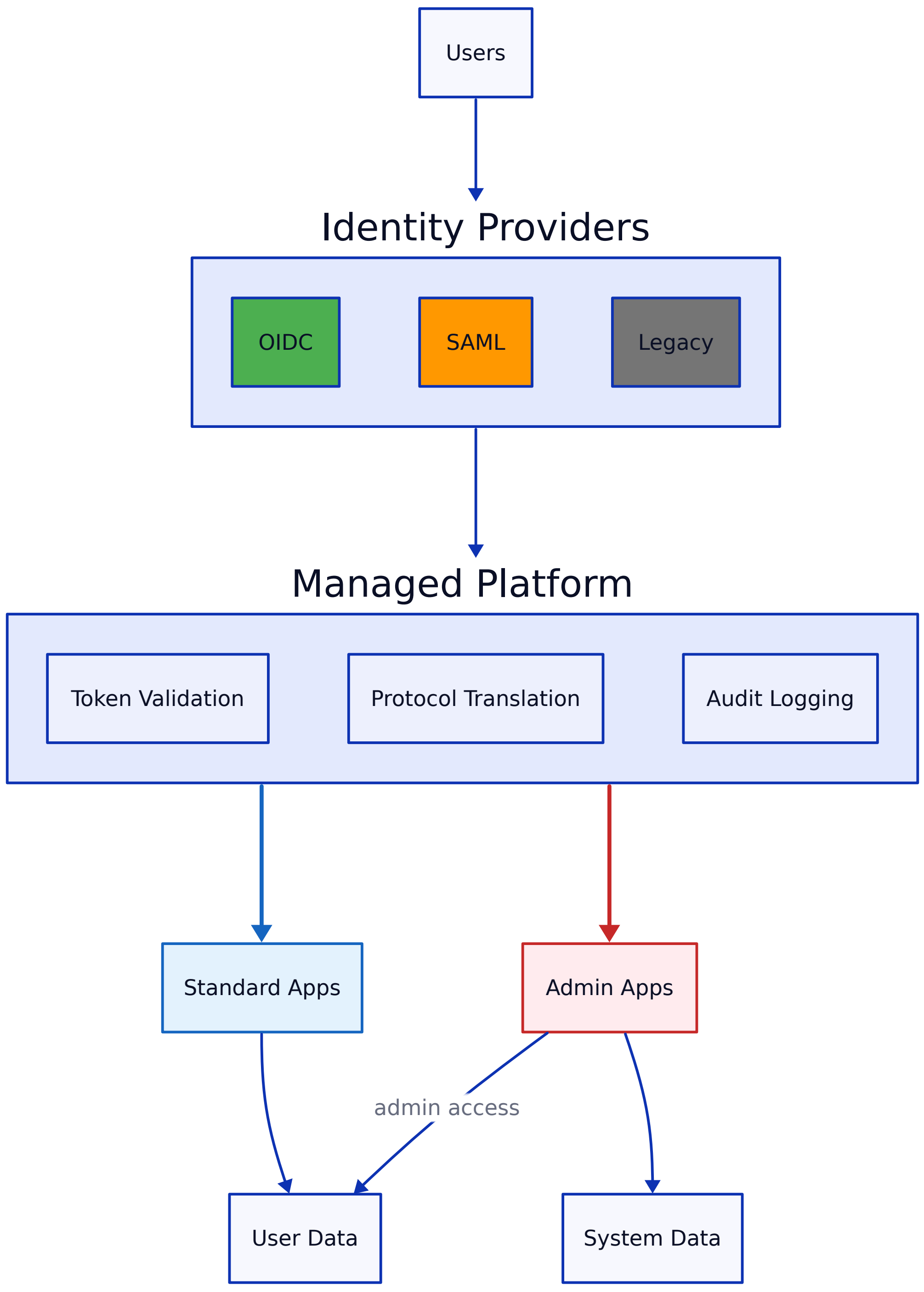
Protocol Standards:

- **Primary**: OpenID Connect for modern identity providers and new integrations
- **Legacy Support**: SAML 2.0 only when upstream providers require it and OIDC is unavailable
- **Security**: Implement PKCE for OIDC public clients and proper token validation
- **Compliance**: Support Digital ID Act 2024 requirements for jurisdiction identity services

Architecture Requirements:

- Applications must integrate through managed identity platforms, not directly with identity providers
- Separate privileged and standard user domains for clear administrative access isolation
- Support multiple upstream identity providers per application
- Maintain audit trails distinguishing privileged from standard user activities

Identity Federation Flow:



Emerging Standards:

- Client applications can leverage emerging OIDC standards around verifiable credentials to simplify adoption of federated identity with jurisdiction providers

Consequences

Benefits:

- Consistent modern federation standard across all applications
- Better security through OIDC's improved token handling and PKCE support
- Simplified integration with jurisdiction citizen identity services
- Clear separation of administrative and standard user access

Risks:

- Legacy systems may require SAML-to-OIDC translation overhead
- Dependency on external identity provider availability
- Additional complexity from managed identity platform requirements

Mitigation:

- Implement fallback authentication mechanisms for critical systems
- Choose identity platforms with high availability and data export capabilities
- Maintain audit trails following ADR 007: Centralized Security Logging

ADR 016: Web Application Edge Protection

Status: Proposed | **Date:** 2025-07-22

Context

Government web applications face heightened security threats including state-sponsored attacks, DDoS campaigns by activist groups, and sophisticated application-layer exploits targeting public services. These attacks can disrupt critical citizen services and damage public trust.

Traditional perimeter security is insufficient for protecting modern web applications that serve millions of citizens. Edge protection through CDNs and WAFs provides the first line of defense, filtering malicious traffic before it reaches application infrastructure.

References:

- ACSC Information Security Manual (ISM)
- ACSC Guidelines for System Hardening
- OWASP Web Application Security Testing Guide

Decision

All public web applications and APIs must use CDN with integrated WAF protection:

CDN Requirements:

- Geographic distribution with SSL/TLS termination at edge
- Cache optimization and origin shielding
- IPv6 dual-stack support

WAF Protection:

- OWASP Top 10 protection rules enabled
- Layer 7 DDoS protection and rate limiting
- Geo-blocking and bot management
- Custom rules for application-specific threats

DDoS Protection:

- AWS Shield Advanced or equivalent

- Real-time attack monitoring and alerting
- DDoS Response Team access

Implementation:

- WAF logs integrated with SIEM systems
- Fail-secure configuration (no fail-open)
- Regular penetration testing and rule tuning
- CI/CD integration for automated deployments

Consequences

Risks of not implementing:

- Service disruption from DDoS attacks
- Exploitation of web application vulnerabilities
- Poor performance and user experience

Benefits:

- Significant reduction in successful attacks
- Improved application performance and availability
- Enhanced security posture at network edge

Operations ADRs

ADR 002: AWS EKS for Cloud Workloads

Status: Accepted | **Date:** 2025-02-17

Context

The challenge is to efficiently manage and scale bespoke workloads in a secure and scalable manner. Traditional server management can be cumbersome and inefficient for dynamic workloads. Provider specific control planes can result in lock-in and artificial constraints limiting technology options.

- Cloud Native Computing Foundation (CNCF) Cloud Native Landscape
- AWS EKS Quickstart

Decision

To address these challenges, we propose using a CNCF Certified Kubernetes platform with automatically managed infrastructure resources. Due to hyperscaler availability and size this would be AWS EKS (Elastic Kubernetes Service) in auto mode. This approach leverages Kubernetes for orchestration, AWS EKS for managed Kubernetes services, AWS Elastic Block Store (EBS) for storage and AWS load balancers for traffic management.

- **AWS EKS Auto Mode**: Provide a managed Kubernetes service, that automatically scales the infrastructure based on workload demands.
- **Managed Storage and NodePools**: Ensure that the underlying infrastructure is maintained and updated by AWS.
- **Load Balancers**: Standardise ingress and traffic management.

Consequences

If we do not adopt this approach, we risk the following:

- **Inefficient Resource Usage**: Manual scaling and management can lead to over-provisioning or under-provisioning of resources.
- **Increased Operational Overhead**: Managing bespoke workload tooling or kubernetes clusters and underlying infrastructure manually can be time-consuming and error-prone.
- **Security Risks**: Operating system, network and workload orchestration updates and patches can lead to security vulnerabilities if not managed.
- **Reduced Availability**: Self managing storage and nodepools , the system may not handle traffic spikes, leading to downtime.

ADR 006: Automated Policy Enforcement

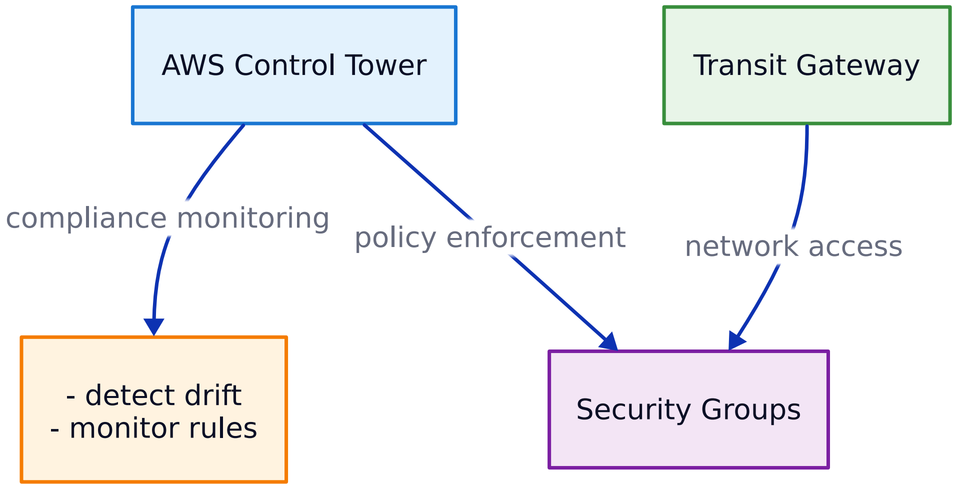
Status: Proposed | **Date:** 2025-07-29

Context

Cloud infrastructure requires automated policy enforcement to prevent misconfigurations, ensure compliance, and provide secure network access patterns. Manual checking cannot scale effectively across multiple accounts and services.

Decision

Implement comprehensive automated policy enforcement using AWS native services for governance, network security, and access control.



Governance Foundation

- **AWS Control Tower**: Account factory, guardrails, and compliance monitoring across organisation
- **Service Control Policies**: Preventive controls blocking non-compliant resource creation
- **AWS Config Rules**: Detective controls for compliance monitoring and drift detection

Network Security & Access

- **Transit Gateway**: Central hub for intra-account resource exposure via security groups
- **Security Group References**: Use security group IDs instead of hardcoded IP addresses for dynamic, maintainable access policies
- **Shield Advanced**: DDoS protection and egress intrusion detection for public-facing resources
- **VPC Flow Logs**: Complete egress traffic monitoring and analysis per WA SOC Cyber Network Management Guideline

Note: This approach creates dependency on AWS for traffic and network protection. Open-source equivalents include Security Onion for network security monitoring, OPNsense and pfSense for firewall and intrusion detection capabilities.

Core Policy Areas

- **Encryption**: Mandatory encryption for all data stores and communications
- **Access Control**: IAM least-privilege access and security group-based resource access
- **Resource Tagging**: Governance and cost allocation requirements
- **Data Sovereignty**: Geographic restrictions for jurisdiction compliance
- **Network Segmentation**: Security group-based micro-segmentation over IP-based rules

Consequences

Benefits:

- Proactive prevention of security misconfigurations through Control Tower guardrails
- Complete egress traffic visibility and monitoring capabilities
- Dynamic, maintainable access policies using security group references
- Centralised network access management via Transit Gateway
- Automated compliance with jurisdiction requirements
- DDoS protection for critical public resources

Risks:

- Dependency on AWS native services for policy enforcement
- Complexity in multi-account Transit Gateway routing
- Potential performance impact from comprehensive logging

Mitigation:

- Implement policy validation in CI/CD pipelines following ADR 010: Infrastructure as Code
- Use security group references over hardcoded IPs for maintainable policies
- Monitor VPC Flow Logs for egress traffic analysis and anomaly detection

ADR 007: Centralised Security Logging

Status: Accepted | **Date:** 2025-02-25

Context

Security logs should be centrally collected to support monitoring, detection, and response capabilities across workloads. Sensitive information logging must minimize to follow data protection regulations and reduce the risk of data breaches. Audit and authentication logs are critical for security monitoring and should collect by default.

- Open Web Application Security Project (OWASP) Logging Cheat Sheet
- Australian Cyber Security Centre (ACSC) Guidelines for system monitoring
- DGOV Technical Baseline for Detection Coverage (MITRE ATT&CK)

Decision

Use centralized logging using Microsoft Sentinel and Amazon CloudWatch.

- Configure default collection for audit and authentication logs to simplify security investigations.
- Container workloads should configure Container insights with enhanced observability + EKS control plane logging of audit and authentication logs by default.
- Logging should configure to avoid capturing and exposing Personally Identifiable Information (PII).
- Review and update logging configurations to ensure coverage and privacy requirements meet.
- Log information used during an investigation should extract and archive to an appropriate location (in alignment with record keeping requirements).

Consequences

Risks of not implementing:

- Decentralized logs may lead to delayed detection and response to security incidents.
- Increased risk of sensitive information exposure leading to potential data breaches and non-compliance with regulations.
- Incomplete audit trails may hinder forensic investigations and compliance audits.

Benefits:

- Improved incident detection and response times.
- Simplified compliance with data protection regulations.
- Centralized management of security logs, reducing operational overhead.

ADR 010: Infrastructure as Code

Status: Accepted | **Date:** 2025-03-10

Context

All environments must be reproducible from source to minimize drift and security risk. Manual changes and missing version control create deployment failures and vulnerabilities.

Compliance Requirements:

- OWASP IaC Security
- ACSC System Hardening

Decision

Golden Path

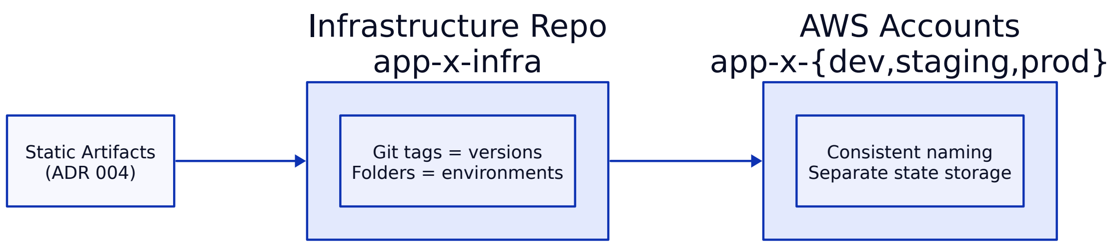
1. **Git Repository Structure**: Single repo per application with `environments/{dev,staging,prod}` folders matching AWS account names (e.g., `app-a-infra` repo → `app-a-dev`, `app-a-`

`staging`, `app-a-prod` accounts)
2. **State Management**: Terraform remote state with locking, separate state per environment
3. **CI Pipeline**:
   - **Validate**: Trivy scan + `terraform plan/kubectl diff` drift check
   - **Plan**: Show proposed changes on PR
   - **Apply**: Deploy on tagged release only
4. **Versioning**: Git tags = semantic versions (x.y.z) deployable to any environment
5. **Disaster Recovery**: Checkout tag + run `just deploy --env=prod` with static artifacts from ADR 004

Required Tools & Practices

Tool	Purpose	Stage	Mandatory
Trivy	Vulnerability scanning	Validate	Yes
Terraform or kubectl/kustomize	Configuration management	Deploy	Yes
Justfiles	Task automation	All	Recommended
devcontainer-base	Dev environment	Local	Recommended
k3d	Local testing	Dev	Optional

Infrastructure as Code Workflow:



Consequences

Without this approach: Configuration drift, security vulnerabilities, failed rollbacks, and inconsistent environments.

With this approach: Secure, reproducible deployments with reliable disaster recovery and automated drift prevention.

References

- ADR 001: Application Isolation
- ADR 002: AWS EKS for Cloud Workloads
- ADR 005: Secrets Management

ADR 014: Object Storage Backups

Status: Proposed | **Date:** 2025-07-22

Context

Current backup approaches lack cross-region redundancy and automated lifecycle management, creating single points of failure and compliance risks for government data retention requirements. Traditional storage systems do not provide the durability and geographic distribution needed for critical government systems.

Key challenges:

- Single region backup storage creating vulnerability to regional outages
- Manual backup processes prone to human error
- Lack of automated recovery testing
- Insufficient geographic separation for disaster recovery

References:

- ACSC Information Security Manual (ISM)
- AWS Well-Architected Framework - Reliability Pillar

Decision

Implement standardized object storage backup solution with automated cross-region replication and lifecycle management for all critical systems and data.

Storage Requirements:

- Object storage with versioning and immutable storage capabilities
- Database, application data, and infrastructure configuration backups
- Encryption at rest and in transit per ADR 005: Secrets Management
- Access controls aligned with ADR 001: Application Isolation

Critical Systems Definition:

- Production databases containing citizen or business data
- Application source code and deployment configurations
- Security logs and audit trails
- Infrastructure as Code templates and state files

Geographic Distribution:

- Cross-region replication within Australia
- Multi-cloud backup for critical systems
- Monitoring integration per ADR 007: Centralised Security Logging

Lifecycle Management:

- Automated storage tiering based on age and access patterns
- Compliance-based retention policies
- Recovery testing and validation procedures

Recovery Objectives:

- **Recovery Time Objective (RTO)**: 4 hours for critical systems, 24 hours for standard systems
- **Recovery Point Objective (RPO)**: 1 hour for databases, 24 hours for static content
- **Implementation Example**: AWS S3 Cross-Region Replication to Australian regions

Consequences

Risks of not implementing:

- Permanent data loss from single point of failure
- Extended recovery times during disasters
- Compliance violations from inadequate data retention
- Regional outages affecting backup availability

Benefits of implementation:

- Automated recovery capabilities meeting defined RTO/RPO objectives
- Compliance with government data retention requirements
- Geographic redundancy protecting against regional disasters
- Reduced operational overhead through automated lifecycle management

ADR 015: Data Governance Standards

Status: Proposed | **Date:** 2025-07-28

Context

Data transformation pipelines require basic governance to ensure quality and compliance. SQLMesh provides built-in capabilities for contracts and lineage that reduce governance overhead.

- SQLMesh Documentation
- Australian Government Data Governance Framework

Decision

Use SQLMesh's semantic understanding for automated data governance with git-based workflows.

Priority Focus Areas

- **Data Contracts**: SQLMesh model definitions serve as schema contracts with automatic breaking change detection
- **Column-Level Lineage**: Automatic lineage tracking through SQLMesh semantic analysis
- **Quality Validation**: SQLMesh unit tests and data diff for transformation validation
- **Audit Integration**: Follow ADR 007: Centralized Security Logging for transformation logs

Consequences

Benefits

- **Automated Governance**: SQLMesh reduces manual governance overhead
- **Git-Based Workflow**: Familiar version control approach
- **Built-in Validation**: Immediate feedback on quality and contract violations

Trade-offs

- **SQLMesh Dependency**: Governance tied to SQLMesh framework
- **Learning Curve**: Teams need SQLMesh knowledge

ADR 017: Analytics Tooling Standards

Status: Proposed | **Date:** 2025-07-28

Context

Organisations need simple, secure reporting that avoids complex JavaScript toolchains. Static reports reduce maintenance and security overhead.

- Quarto Documentation
- Evidence BI Documentation
- Government Digital Service Design Standards

Decision

Use Quarto for static reporting with Evidence BI for high-interactivity cases only.

Primary: Quarto Framework

- **Static Reports**: Markdown-based with embedded visualizations
- **Multi-format**: HTML, PDF outputs from single source
- **Git Integration**: Reports alongside code in version control
- **Compliance Ready**: Accessibility and professional formatting

Secondary: Evidence BI

- **Limited Use**: Only useful for interactive drilldown reporting (simpler to host than e.g. PowerBI)
- **SQL-based**: Minimal JavaScript complexity

Integration

- **Data Sources**: Connect via ADR 018: Database Patterns
- **Deployment**: Use ADR 016: Web Application Edge Protection for secure distribution

Consequences

Benefits

- **Low Maintenance**: Static reports reduce operational overhead
- **Security**: Reduced attack surface vs dynamic dashboards
- **Git Integration**: Version-controlled reports ensure consistency

Trade-offs

- **Limited Interactivity**: Static reports lack real-time exploration
- **Learning Curve**: Teams need Quarto knowledge

ADR 018: Database Patterns

Status: Proposed | **Date:** 2025-07-28

Context

Applications need managed databases with automatic scaling and jurisdiction-compliant backup strategies.

- AWS Aurora Serverless v2 Documentation
- Percona Everest Documentation and Pigsty Documentation for development/non-AWS environments

Decision

Use Aurora Serverless v2 outside EKS clusters with automated scaling, multi-AZ deployment, and dual backup strategy.

Implementation

- **Database**: Aurora Serverless v2 (PostgreSQL/MySQL) with built-in connection pooling and automatic scaling
- **Deployment**: Outside EKS cluster (handles complexity automatically)
- **Credentials**: Follow ADR 005: Secrets Management for endpoint and credential management
- **Backup**: Follow ADR 014: Object Storage Backups plus AWS automated snapshots
- **Security**: Follow ADR 007: Centralized Security Logging and ADR 012: Privileged Remote Access

Consequences

Benefits

- **Cost Efficiency**: Serverless scaling reduces costs during low usage
- **Low Maintenance**: Managed service with automatic scaling and high availability
- **Compliance**: Dual backup strategy meets jurisdiction requirements

Trade-offs

- **Vendor Lock-in**: AWS-specific (consider Percona Everest or Pigsty for development/non-AWS)
- **Cold Start**: Brief delays when scaling from zero

Development ADRs

ADR 003: API Documentation Standards

Status: Accepted | **Date:** 2025-03-26

Context

Secure, maintainable APIs require mature frameworks with low complexity and industry standard compliance. Where existing standards exist, prefer them over bespoke REST APIs.

Compliance Requirements:

- ACSC Software Development Guidelines
- OWASP API Security
- OpenAPI Specification

Decision

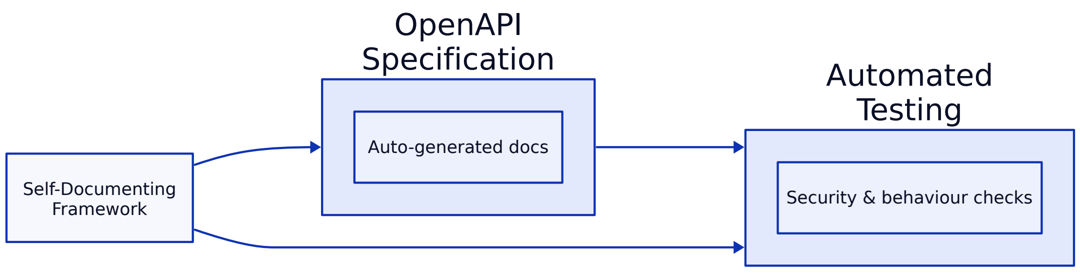
API Requirements

Requirement	Standard	Mandatory
Documentation	OpenAPI Specification	Yes
Testing	Restish CLI scripts	Yes
Framework	Huma or FastAPI	Recommended
Naming	Consistent convention	Yes
Security	OWASP API security coverage	Yes
Exposure	No admin APIs on Internet	Yes

Development Guidelines

- **Self-Documenting**: Use frameworks that auto-generate OpenAPI specs
- **Data Types**: Prefer standard types over custom formats
- **Segregation**: Separate APIs by purpose
- **Testing**: Include security vulnerability checks in test scripts

API Development Flow:



Consequences

Without this approach: Security vulnerabilities, documentation drift, poor quality APIs, and increased maintenance overhead.

With this approach: Secure, well-documented APIs with automated testing and reduced maintenance burden.

ADR 004: CI/CD Quality Assurance

Status: Accepted | **Date:** 2025-03-10

Context

Ensure security and integrity of software artifacts that are consumed by infrastructure repositories per ADR 010. Threat actors exploit vulnerabilities in code, dependencies, container images, and exposed secrets.

Compliance Requirements:

- OWASP CI/CD Security
- ACSC Software Development Guidelines

Decision

CI/CD Pipeline Requirements

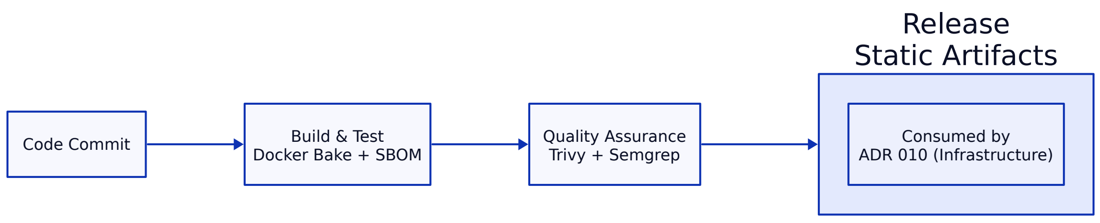
Pipeline Flow: Code Commit → Build & Test → Quality Assurance → Release

Stage	Tools	Purpose	Mandatory
Build	Docker Bake	Multi-platform builds with SBOM/provenance	Yes
Scan	Trivy	Vulnerability scanning	Yes
Analysis	Semgrep	Static code analysis	Yes
Test	Playwright	End-to-end testing	Recommended
Performance	Grafana K6	Load testing	Optional
API	Restish	API validation per ADR 003	Optional

Development Environment

- Use devcontainer-base for standardized tooling
- Use Justfiles for task automation
- Use GitHub Actions for CI/CD automation

CI/CD Pipeline:



Consequences

Without this approach: Vulnerable containers deployed, exposed secrets, compromised application integrity, and compliance violations.

With this approach: Secure, tested artifacts with automated vulnerability remediation and compliance alignment.

References

- ADR 003: API Documentation Standards
- ADR 010: Infrastructure as Code

ADR 009: Release Documentation Standards

Status: Accepted | **Date:** 2025-03-04

Context

To ensure clear communication of changes and updates to security and infrastructure operations teams, release notes should standardize. The release notes should succinctly capture key information, including new features, improvements, bug fixes, security updates, and infrastructure changes, with links to relevant changelogs.

- Australian Cyber Security Centre (ACSC) Guidelines for Software Development

Decision

Adopt a standardised release notes template in Markdown format. Brief descriptions should include the security implications and operational impacts of changes such as vulnerability fixes, compliance improvements, or changes to authentication and authorization mechanisms. Descriptions should also detail operational aspects, including deployment processes, logging & monitoring considerations, and any modifications to Infrastructure as Code (IaC).

Git Tagging Requirements:

- Create a git tag for each release following semantic versioning (v1.0.0, v1.1.0, etc.)
- Tags must annotate with release notes summary
- Tags should create after all ADR acceptance and README updates
- Tag message should reference the release documentation

A template provide below that can tailor per project. A completed release notes Markdown document should provide with all proposed changes.

```
## Release Notes

### Overview
- **Name:** Name
- **Version:** [Version Number](#)
- **Previous Version:** [Version Number](#)

### Changes and Testing

High level summary

**New Features & Improvements**:
- [Feature/Improvement 1]: Brief description including testing.
- [Feature/Improvement 2]: Brief description including testing.

**Bug Fixes & Security Updates**:
- [Bug Fix/Security Update 1]: Brief description with severity level and
- [Bug Fix/Security Update 2]: Brief description with severity level and
- **Response Timelines**: Critical (24h), High (7d), Medium (30d), Low (

### Changelogs

*Only include list items changed by this release*

- **Code**: Brief description. [View Changes](#)
- **Infrastructure**: Brief description. [View Changes](#)
- **Configuration & Secrets**: Brief description.

### Known Issues
- [Known Issue 1]: Brief description.
- [Known Issue 2]: Brief description.

### Action Required
- [Action 1]: Brief description of any action required by users or stake
- [Action 2]: Brief description of any action required by users or stake

### Contact
For any questions or issues, please contact [Contact Information].
```

Consequences

Risks of not implementing:

- Miscommunication between teams.
- Loss of context leading to weakened decision making procedures.

Positive Consequences:

- Improved clarity and consistency in communicating release updates.
- Consistent tracking of changes and updates for security and infrastructure teams.
- Enhanced collaboration and understanding across teams.

Reference Architectures

Reference Architecture: Content Management

Status: Proposed | **Date:** 2025-07-28

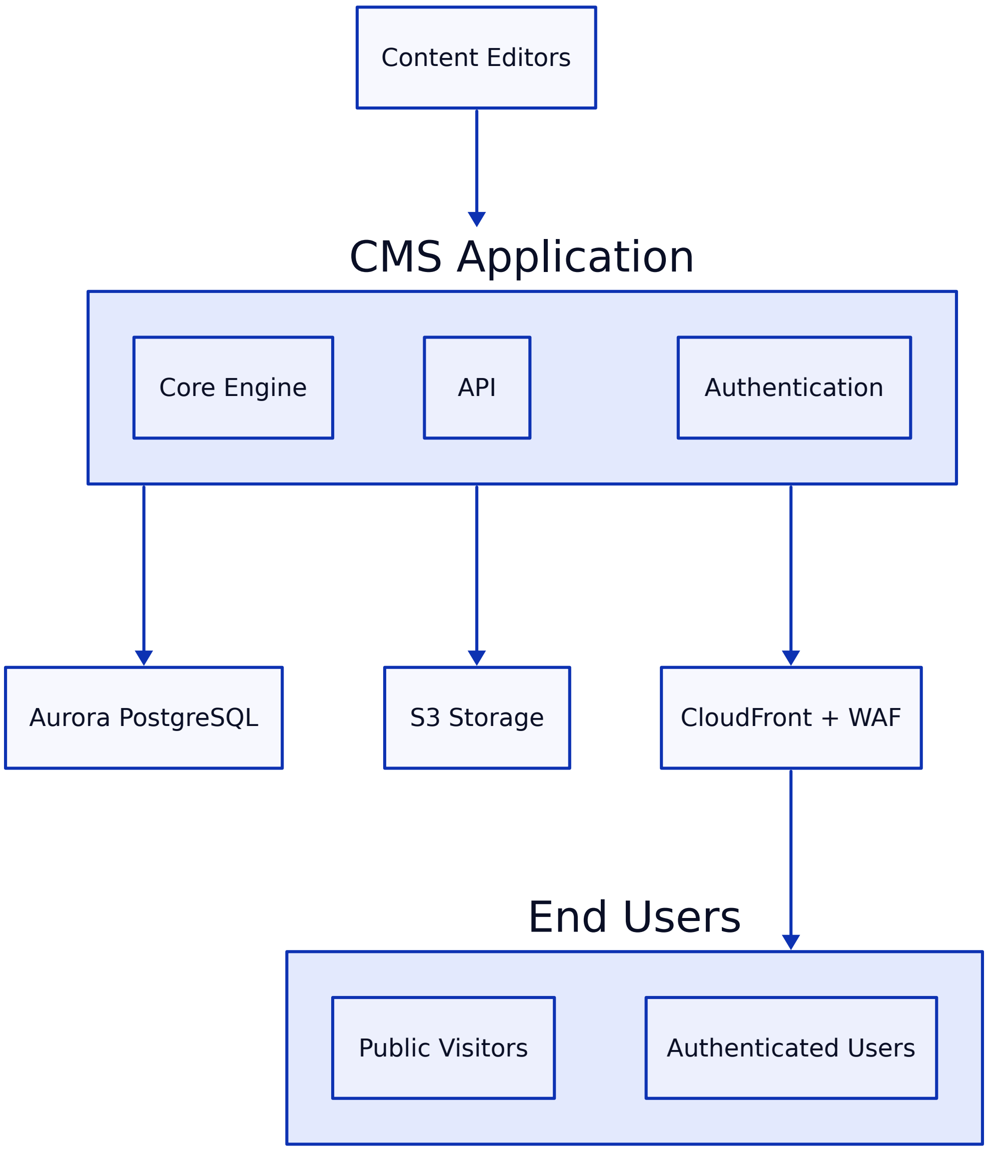
When to Use This Pattern

Use when building websites, content portals, or applications requiring structured content management with editorial workflows.

Overview

Template for implementing content management systems to meet jurisdiction compliance requirements.

Core Components



Project Kickoff Steps

Foundation Setup

1. **Apply Isolation** - Follow ADR 001: Application Isolation for CMS service network and runtime separation
2. **Deploy Infrastructure** - Follow ADR 002: AWS EKS for Cloud Workloads for CMS container deployment
3. **Configure Infrastructure** - Follow ADR 010: Infrastructure as Code for reproducible deployments
4. **Setup Database** - Follow ADR 018: Database Patterns for Aurora Serverless v2 content storage

Security & Operations

1. **Configure Secrets Management** - Follow ADR 005: Secrets Management for database credentials and API keys
2. **Setup Logging** - Follow ADR 007: Centralized Security Logging for audit trails and editorial tracking
3. **Setup Backup Strategy** - Follow ADR 014: Object Storage Backups for content and media backup
4. **Configure Edge Protection** - Follow ADR 016: Web Application Edge Protection for CDN and WAF setup
5. **Identity Integration** - Follow ADR 012: Privileged Remote Access for editorial authentication

Implementation Details

Content Workflows & Editorial:

- Configure content workflows and editorial approval processes
- Setup media asset management and CDN integration per ADR 016: Web Application Edge Protection
- Implement headless CMS API following ADR 003: API Documentation Standards
- Configure content moderation and approval workflows

Compliance & Accessibility:

- Configure WCAG 2.1 AA accessibility compliance and automated testing
- Setup jurisdiction-specific compliance requirements (e.g., privacy policies, cookie consent)
- Implement content governance and retention policies per ADR 015: Data Governance Standards
- Configure multilingual content management if required

Performance & SEO:

- Setup SEO metadata management and structured data (JSON-LD)
- Implement content performance monitoring per ADR 007: Centralized Security Logging
- Configure CDN caching strategies and cache invalidation
- Setup content analytics and user behavior tracking

Reference Architecture: Data Pipelines

Status: Proposed | **Date:** 2025-01-28

When to Use This Pattern

Use when building data processing systems for analytics, ETL workloads, or real-time data transformation across organisational systems.

Overview

Template for implementing scalable data pipelines using SQLMesh framework with S3 object storage, Aurora database patterns, and modern analytics engines (DuckDB/DuckLake) or legacy engines (Iceberg/Trino/Athena) based on existing data lake alignment.

Core Components

Data Sources -> SQLMesh -> S3 Storage SQLMesh -> Aurora PostgreSQL S3 Storage -> Analytics Engine Analytics Engine -> Reports

Data Sources: { Databases APIs Files }

SQLMesh: { Orchestration Transformation Quality Checks }

S3 Storage: Raw & Processed Data Aurora PostgreSQL: Metadata & Schema Analytics Engine: DuckDB/Athena

Reports: { Quarto Evidence BI Data API }

Project Kickoff Steps

Foundation Setup

1. **Apply Isolation** - Follow ADR 001: Application Isolation for data processing network and runtime separation
2. **Deploy Infrastructure** - Follow ADR 002: AWS EKS for Cloud Workloads for SQLMesh container deployment
3. **Configure Infrastructure** - Follow ADR 010: Infrastructure as Code for reproducible data infrastructure
4. **Setup Database** - Follow ADR 018: Database Patterns for Aurora Serverless v2 as data warehouse

Security & Operations

1. **Configure Secrets Management** - Follow ADR 005: Secrets Management for data source credentials and API keys
2. **Setup Logging** - Follow ADR 007: Centralized Security Logging for transformation audit trails
3. **Setup Backup Strategy** - Follow ADR 014: Object Storage Backups for data warehouse backup
4. **Data Governance** - Follow ADR 015: Data Governance Standards for SQLMesh data contracts and lineage

Development Process

1. **Configure CI/CD** - Follow ADR 004: CI/CD Quality Assurance for automated testing of data transformations
2. **Setup Release Process** - Follow ADR 009: Release Documentation Standards for SQLMesh model versioning
3. **Analytics Tools** - Follow ADR 017: Analytics Tooling Standards for Quarto and Evidence BI integration

Implementation Details

Data Processing & Quality:

- Configure SQLMesh for data transformation and orchestration
- Setup S3 object storage for data files and Aurora for metadata
- Implement data quality validation rules and testing frameworks
- Configure DuckDB/DuckLake or Iceberg/Trino based on existing data lake alignment

Cost Optimization & Performance:

- Configure Aurora Serverless v2 autoscaling for cost-effective metadata storage
- Implement S3 lifecycle policies for data archival (Intelligent Tiering → Glacier)
- Setup DuckDB for cost-effective analytics vs Athena for large-scale queries
- Configure SQLMesh incremental processing to minimize compute costs

Data Governance & API Access:

- Setup data API following ADR 003: API Documentation Standards
- Implement PII handling, data classification, and retention policies per ADR 015: Data Governance Standards
- Configure data lineage tracking and impact analysis through SQLMesh
- Setup automated data profiling and anomaly detection

Reference Architecture: Identity Management

Status: Proposed | **Date:** 2025-07-29

When to Use This Pattern

Use when building systems requiring federated identity management, single sign-on across multiple services, or integration with jurisdiction identity providers using OIDC standards.

Overview

Template for implementing OIDC-based identity federation with upstream identity providers (verifiable credentials, Australian Government Digital ID) and downstream identity consumers (AWS Cognito, Microsoft Entra ID) for comprehensive identity management across organisational boundaries.

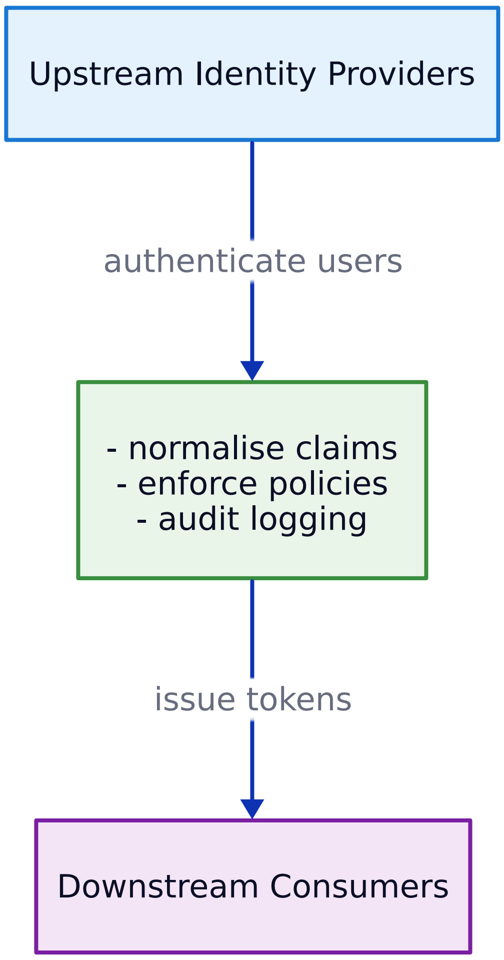
Identity Federation Pattern

This pattern implements a **broker-based identity federation** that translates between upstream identity providers (Government Digital ID, verifiable credentials) and downstream identity consumers (AWS Cognito, Microsoft Entra ID).

Key Benefits:

- Single integration point for multiple upstream providers
- Standardised OIDC/SAML interface for downstream consumers
- Centralised policy enforcement and audit logging
- Support for both government and commercial identity ecosystems

Core Components



The architecture consists of three main layers:

Upstream Identity Providers supply user identities through standards-based protocols:

- Government Digital ID for Australian citizens
- Verifiable Credentials providers for professional credentials
- External OIDC providers for commercial identity systems

Identity Broker acts as the central translation layer that:

- Accepts authentication from multiple upstream providers
- Normalises identity claims and attributes across providers
- Issues standardised tokens to downstream consumers

Downstream Identity Consumers consume the normalised identity tokens:

- AWS Cognito for cloud-native applications

- Microsoft Entra ID for enterprise applications
- Custom applications using OIDC/SAML protocols

Project Kickoff Steps

1. **Infrastructure Foundation** - Follow ADR 001: Application Isolation, ADR 002: AWS EKS for Cloud Workloads, and ADR 018: Database Patterns for identity service deployment and data separation
2. **Security & Secrets** - Follow ADR 005: Secrets Management for OIDC client secrets and ADR 007: Centralized Security Logging for authentication audit trails
3. **Identity Federation** - Follow ADR 013: Identity Federation Standards for upstream provider integration and downstream consumer configuration
4. **Privileged Administration** - Follow ADR 012: Privileged Remote Access for identity service administration access

Implementation Considerations

Privacy & PII Protection (Digital ID Act 2024):

- **Data minimisation**: Prohibit collection beyond identity verification requirements
- **No single identifiers**: Prevent tracking across services using persistent identifiers
- **Marketing restrictions**: Prohibit disclosure of identity information for marketing purposes
- **Voluntary participation**: Users cannot be required to create Digital ID for service access
- **Biometric safeguards**: Restrict collection, use, and disclosure of biometric information
- **Breach notification**: Implement cyber security and fraud incident management processes

Identity Proofing Level Selection:

- **IP1-IP2**: Low-risk transactions with minimal personal information exposure
- **IP2+**: Higher-risk services requiring biometric verification and stronger assurance
- **Risk assessment**: Match proofing level to transaction risk and data sensitivity
- **Credential binding**: Ensure authentication levels align with proofing requirements

Standards Compliance:

- Verifiable credentials: ISO/IEC 18013-5:2021 and W3C Verifiable Credentials
- Government Digital ID: Digital ID Act 2024 privacy and security requirements
- International interoperability: eIDAS Regulation patterns

Reference Architecture: OpenAPI Backend

Status: Proposed | **Date:** 2025-07-28

When to Use This Pattern

Use when you need clear separation between user-facing and administrative operations in REST APIs requiring structured documentation and testing.

Overview

Template for implementing OpenAPI-first API services with complete separation between user-facing and administrative operations.

Core Components

API Clients -> Edge Protection -> Routing

Routing -> Standard APIs: api.domain Routing -> Admin APIs: admin.domain

Standard APIs: { style: { fill: "#e3f2fd" stroke: "#1565c0" } }

Admin APIs: { style: { fill: "#ffebee" stroke: "#c62828" } }

Standard APIs -> User Database Admin APIs -> System Database Admin APIs -> User Database: admin access
Standard APIs (`/api/v1/*`): Business operations for authenticated users

- Examples: `/api/v1/users/profile`, `/api/v1/orders`, `/api/v1/documents`
- Domain: `api.domain` with Standard Realm authentication

Administrative APIs (`/admin/v1/*`): System management for privileged users

- Examples: `/admin/v1/users`, `/admin/v1/system/config`, `/admin/v1/audit-logs`
- Domain: `admin.domain` with Admin Realm authentication

Complete Separation: Two separate routers provide full network and authentication isolation between standard and administrative operations per ADR 013: Identity Federation Standards.

Project Kickoff Steps

1. **Infrastructure Foundation** - Follow ADR 001: Application Isolation and ADR 002: AWS EKS for Cloud Workloads
2. **API Standards** - Follow ADR 003: API Documentation Standards for OpenAPI specification
3. **Identity Federation** - Follow ADR 013: Identity Federation Standards for domain separation
4. **Edge Protection** - Follow ADR 016: Web Application Edge Protection for rate limiting and security
5. **Database & Secrets** - Follow ADR 018: Database Patterns and ADR 005: Secrets Management
6. **Logging & Monitoring** - Follow ADR 007: Centralized Security Logging for audit trails

Templates

ADR ###: Specific Decision Title

Status: Proposed | **Date:** YYYY-MM-DD

Context

What problem are we solving? Include background and constraints.

Decision

What we decided and how to implement it:

- **Requirement 1**: Specific implementation detail
- **Requirement 2**: Configuration specifics
- **Requirement 3**: Monitoring approach

Consequences

Positive:

- Benefit 1 with explanation
- Benefit 2 with explanation

Negative:

- Risk 1 with mitigation
- Risk 2 with mitigation

Reference Architecture: Pattern Name

Status: Proposed | **Date:** YYYY-MM-DD

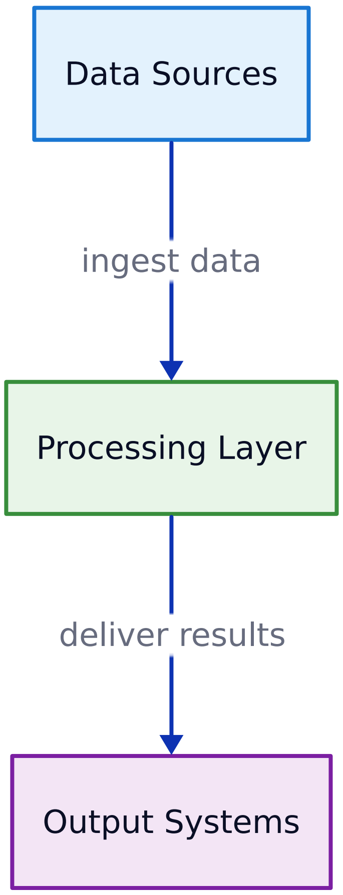
When to Use This Pattern

Clear use case description for when to apply this architecture.

Overview

Brief template description focusing on practical implementation.

Core Components



Project Kickoff Steps

1. **Step Name** - Follow relevant ADRs for implementation
2. **Next Step** - *ADR needed for missing standards*
3. **Final Step** - Reference to existing practices

Contributing Guide

When to Create ADRs

Create ADRs for foundational decisions only:

- High cost to change mid/late project
- Architectural patterns and technology standards
- Security frameworks and compliance requirements

- Infrastructure patterns that affect multiple teams

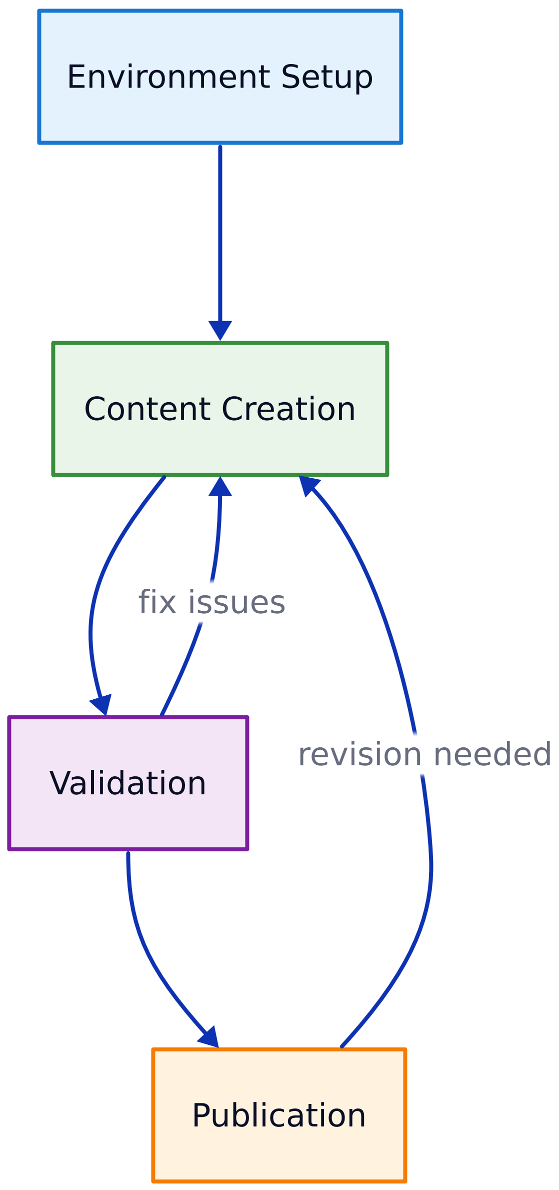
Don't create ADRs for:

- Implementation details (use documentation)
- Project-specific configurations
- Operational procedures that change frequently
- Tool-specific guidance that belongs in user manuals

Quick Workflow

1. **Open in Codespaces** → Automatic tool setup
2. **Get number** → `just next-number`
3. **Create file** → `###-short-name.md` in correct directory (see content types)
4. **Write content** → Follow template below
5. **Lint** → `just lint` to fix formatting, check SUMMARY.md, and validate links
6. **Add to SUMMARY.md** → Include new ADR in navigation (required for mdBook)
7. **Submit PR** → Ready for review

ADR Creation Workflow:



Directory Structure

Directory	Content
`development/`	API standards, CI/CD, releases
`operations/`	Infrastructure, logging, config
`security/`	Isolation, secrets, AI governance
`reference-architectures/`	Project kickoff templates

Content Types: When to Use What

ADRs (Architecture Decision Records)

Purpose: Document foundational technology decisions that are expensive to change
Format: `###-decision-name.md` in `development/`, `operations/`, or `security/`
Examples: “AWS EKS for workloads”, “Secrets management approach”, “API standards”

Reference Architectures

Purpose: Project kickoff templates that combine multiple existing ADRs
Format: `descriptive-name.md` in `reference-architectures/`
Examples: “Content Management”, “Data Pipelines”, “Identity Management”

Rule: Reference architectures should only link to existing ADRs, not create new ones.

ADR Template

See templates/adr-template.md for the complete template.

Note: ADR numbers are globally unique across all directories (gaps from removed drafts are normal)

Reference Architecture Template

See templates/reference-architecture-template.md for the complete template.

Quality Standards

Before submitting:

- ☐ Title is concise (under 50 characters) and actionable
- ☐ All acronyms defined on first use
- ☐ Active voice (not passive)
- ☐ Passes `just lint` without errors

Title Examples:

- GOOD: “ADR 002: AWS EKS for Cloud Workloads” (concise, ~30 chars)
- GOOD: “ADR 008: Email Authentication Protocols” (specific, clear)
- BAD: “ADR 004: Enforce release quality with CI/CD prechecks and build attestation” (too long)
- BAD: “Container stuff” or “Security improvements” (too vague)

Status Guide

Status	Meaning
`Proposed`	Under review
`Accepted`	Active decision
`Superseded`	Replaced by newer ADR

ADR References

Reference format:

- `[ADR 005: Secrets Management](../security/005-secrets-management.md)`
- Quick reference: `per ADR 005`
- Multiple refs: `aligned with ADR 001 and ADR 005`

Examples:

- "Encryption handled per ADR 005: Secrets Management"
- "Access controls aligned with ADR 001"

Writing Tips

- **Be specific**: "Use AWS EKS auto mode" not "Use containers"
- **Include implementation**: How, not just what
- **Define scope**: What's included and excluded
- **Reference standards**: Link to external docs
- **Australian English**: Use "organisation" not "organization", "jurisdiction" not "government"
- **Character usage**: Use plain-text safe Unicode - avoid emoji, smart quotes, em-dashes for PDF compatibility
- **D2 diagrams**: Use D2 format for diagrams with clean syntax and universal compatibility
  - Use when text alone isn't sufficient (system relationships, data flows, workflows)
  - Keep simple: 5-7 components max, clear labels, logical flow, consistent colors

Glossary

Acronyms and Definitions

ACSC - Australian Cyber Security Centre
ADR - Architecture Decision Record
API - Application Programming Interface
ATT&CK - Adversarial Tactics, Techniques & Common Knowledge (MITRE)
AWS - Amazon Web Services
BIMI - Brand Indicators for Message Identification
CDN - Content Delivery Network
CI/CD - Continuous Integration/Continuous Deployment
CNCF - Cloud Native Computing Foundation
DGOV - Office of Digital Government (Western Australia)
DKIM - DomainKeys Identified Mail
DMARC - Domain-based Message Authentication, Reporting and Conformance
DNS - Domain Name System
DTT - Digital Transformation and Technology Unit
EKS - Elastic Kubernetes Service (AWS)
GCP - Google Cloud Platform
IAM - Identity and Access Management
IAP - Identity-Aware Proxy
JIT - Just-In-Time
OWASP - Open Web Application Security Project
RDP - Remote Desktop Protocol
RPO - Recovery Point Objective
RTO - Recovery Time Objective
SIEM - Security Information and Event Management
SPF - Sender Policy Framework
TLS - Transport Layer Security
VPN - Virtual Private Network
WAF - Web Application Firewall