Cloud Security

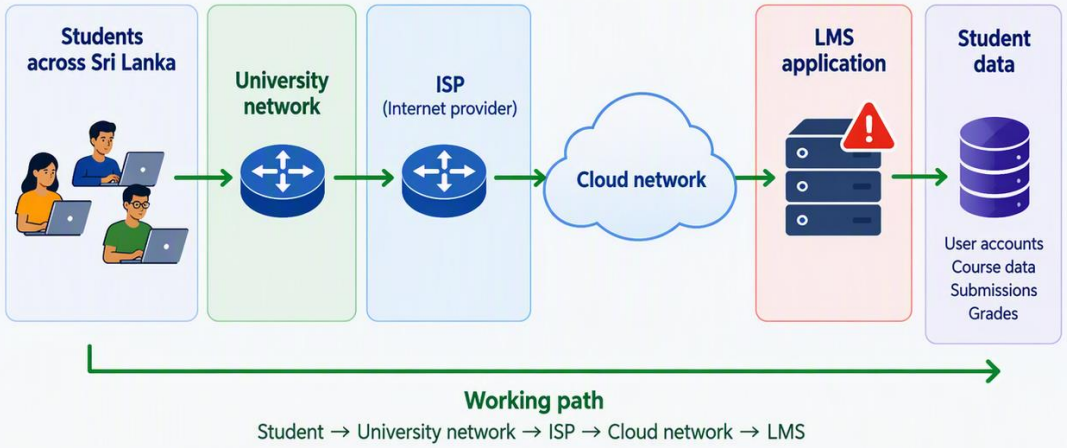


ITN 4252 Advanced Topics in Networking

Partially based on resources from
Cisco, Palo Alto, Snyk, AWS, and Microsoft

Something has changed in the LMS

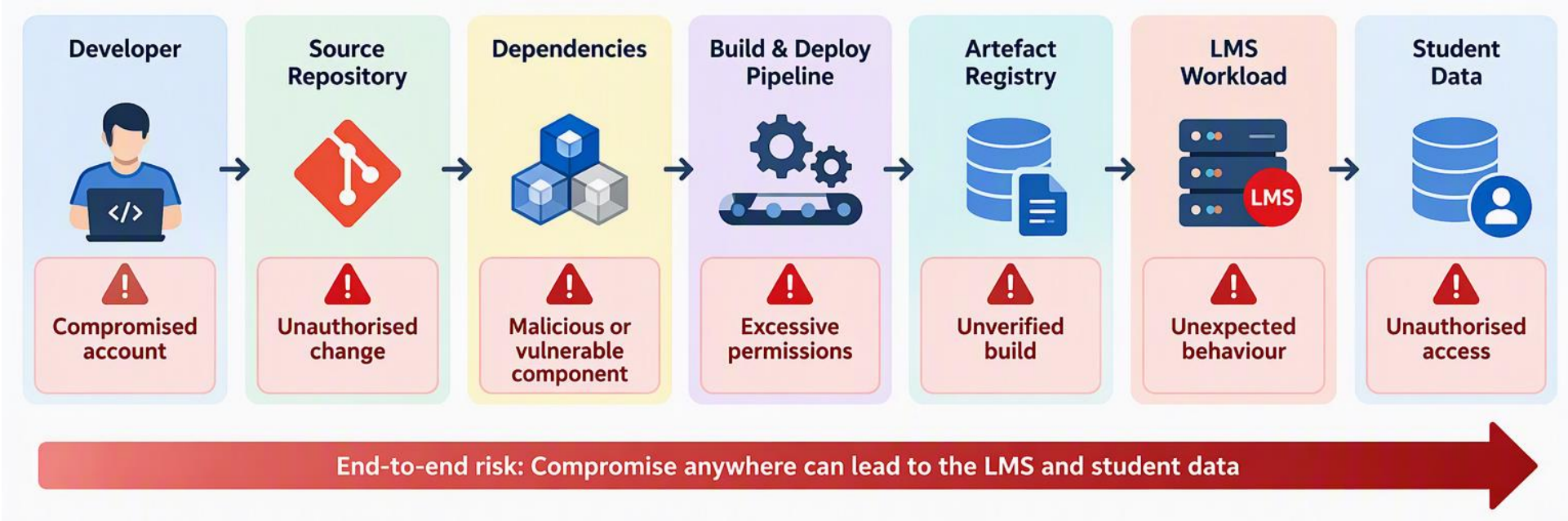
- After the latest release:
  - A new administrator account appears
  - The LMS makes unexpected outbound connections
  - Student data is accessed outside normal patterns
  - The running application does not match the approved build
- **The network is healthy, so what changed?**





Follow the suspected attack path

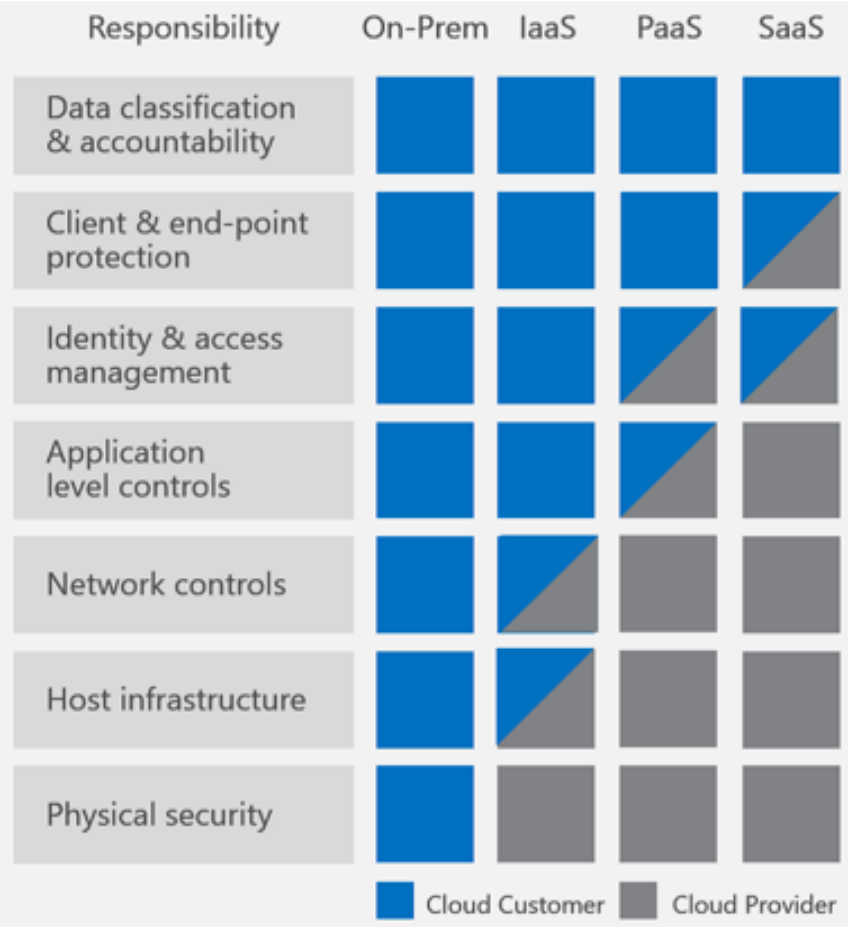
- How could an untrusted change reach student data?



- **Cloud security depends on the trust between every stage.**

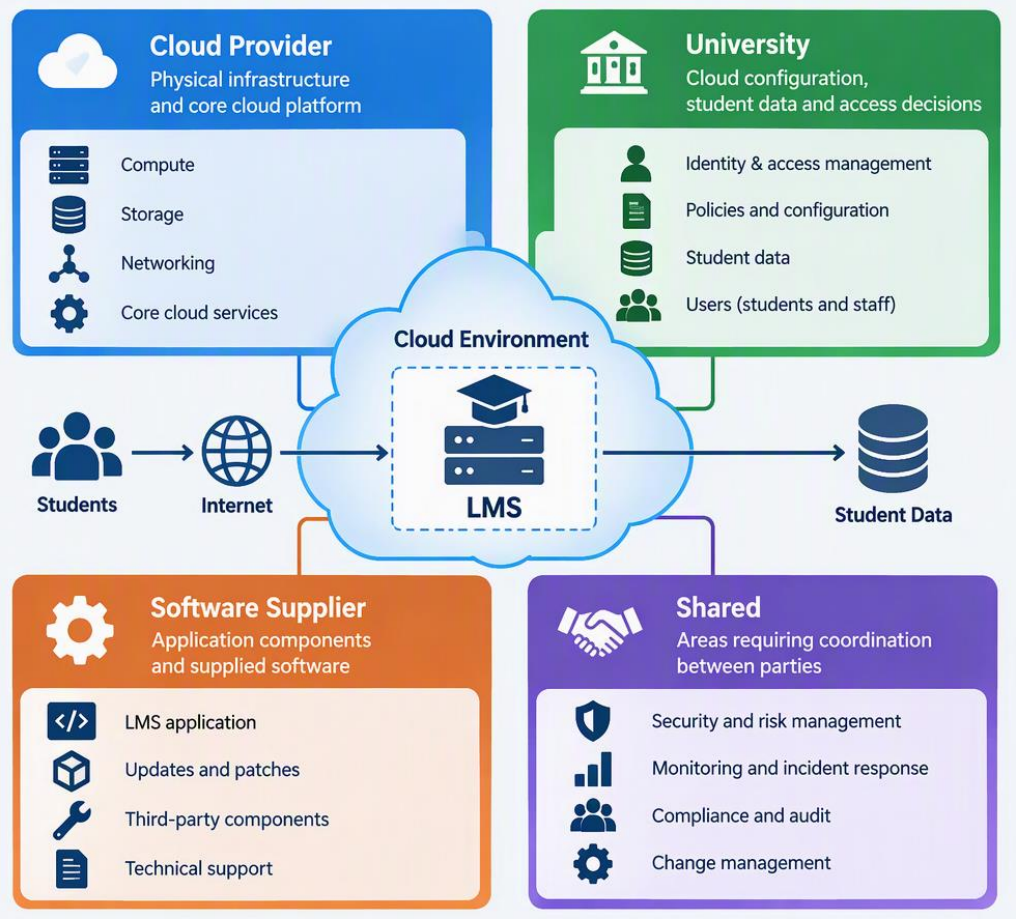
Cloud security is a shared responsibility

- Cloud provider
  - Protects the underlying cloud infrastructure
- University
  - Protects its data, identities, applications and configurations
- Shared
  - Both parties contribute to secure and resilient services
- **Moving to the cloud transfers tasks, not the university's risk.**



Who owns each security control?

- Who is responsible?
- For each control, decide:
  - Cloud provider
  - University
  - Software supplier
  - Shared responsibility
- Security depends on knowing who must act.
- **An unclear responsibility can become an uncontrolled risk.**



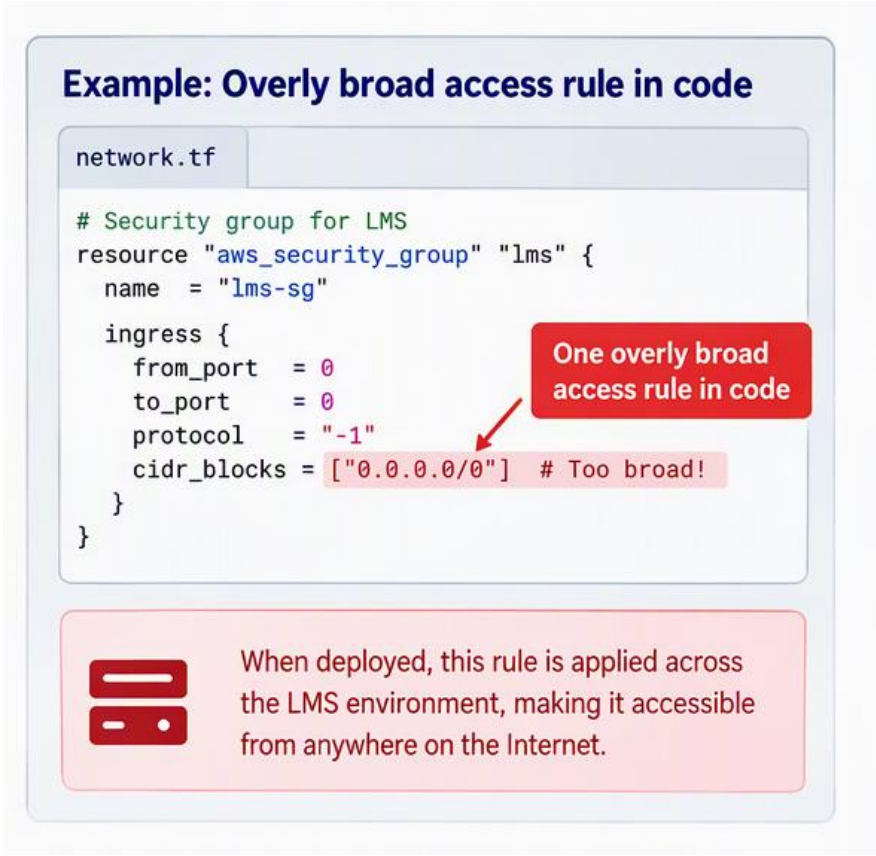
Cloud automation increases the blast radius

- Automation can scale mistakes too.
- Cloud platforms make change fast and repeatable:
  - One configuration can affect many resources
  - Automated deployments reproduce changes consistently
  - Excessive permissions increase potential impact
  - Misconfigurations can spread before they are noticed
- **Automation increases speed, for both good and bad changes.**



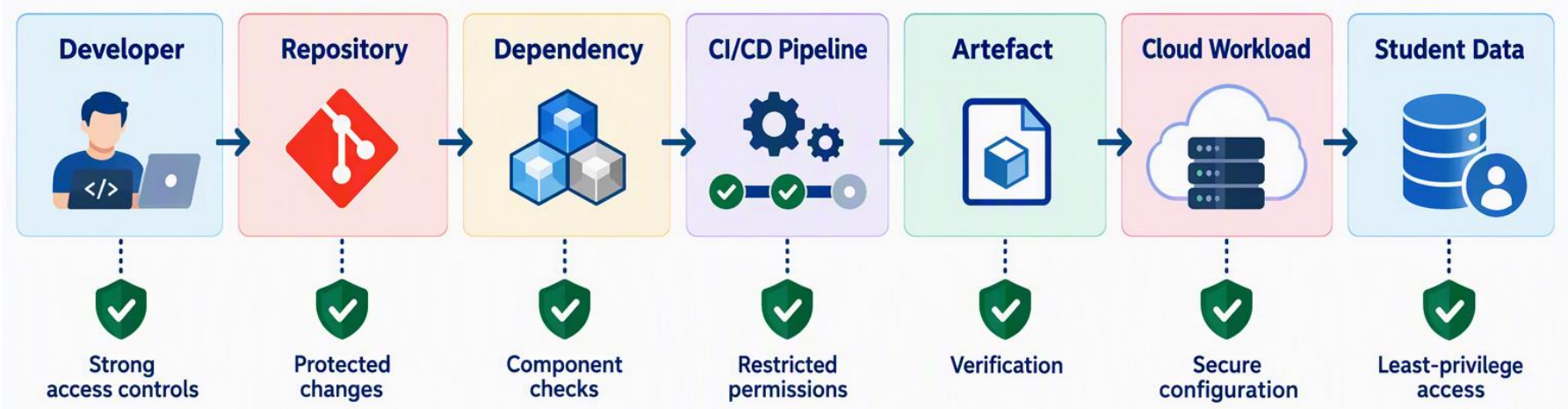
Cloud configuration becomes code

- Infrastructure as Code enables:
  - **Repeatable** cloud environments
  - **Version-controlled** configuration
  - **Reviewable** infrastructure changes
  - **Automated** deployment and validation
  - **Consistent** security controls
- **The same automation that prevents drift can also reproduce a bad configuration.**



Where can we break the attack chain?

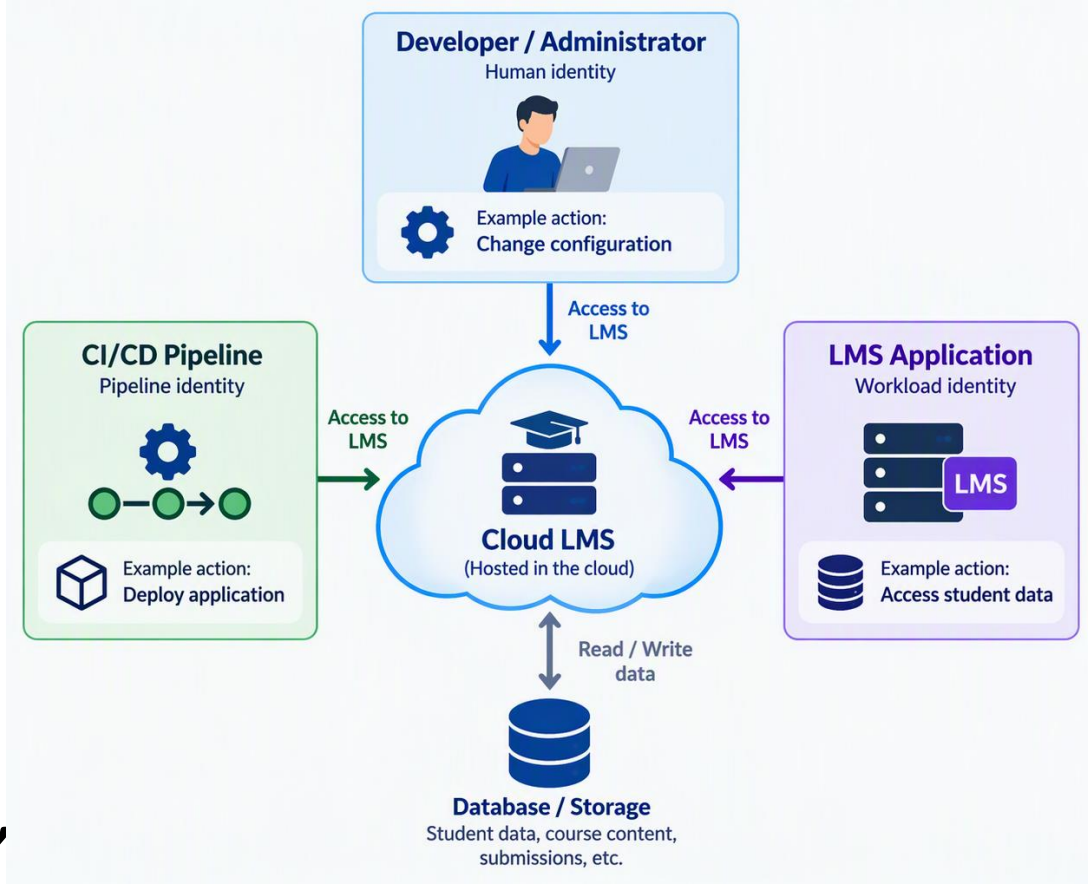
- Every stage is an opportunity to stop the attack.



- For each stage, ask:
  - **Prevent:** What could stop the attack?
  - **Detect:** What would reveal it?
  - **Limit:** What reduces the impact?
- **Cloud security works best when one failed control does not expose the entire service.**

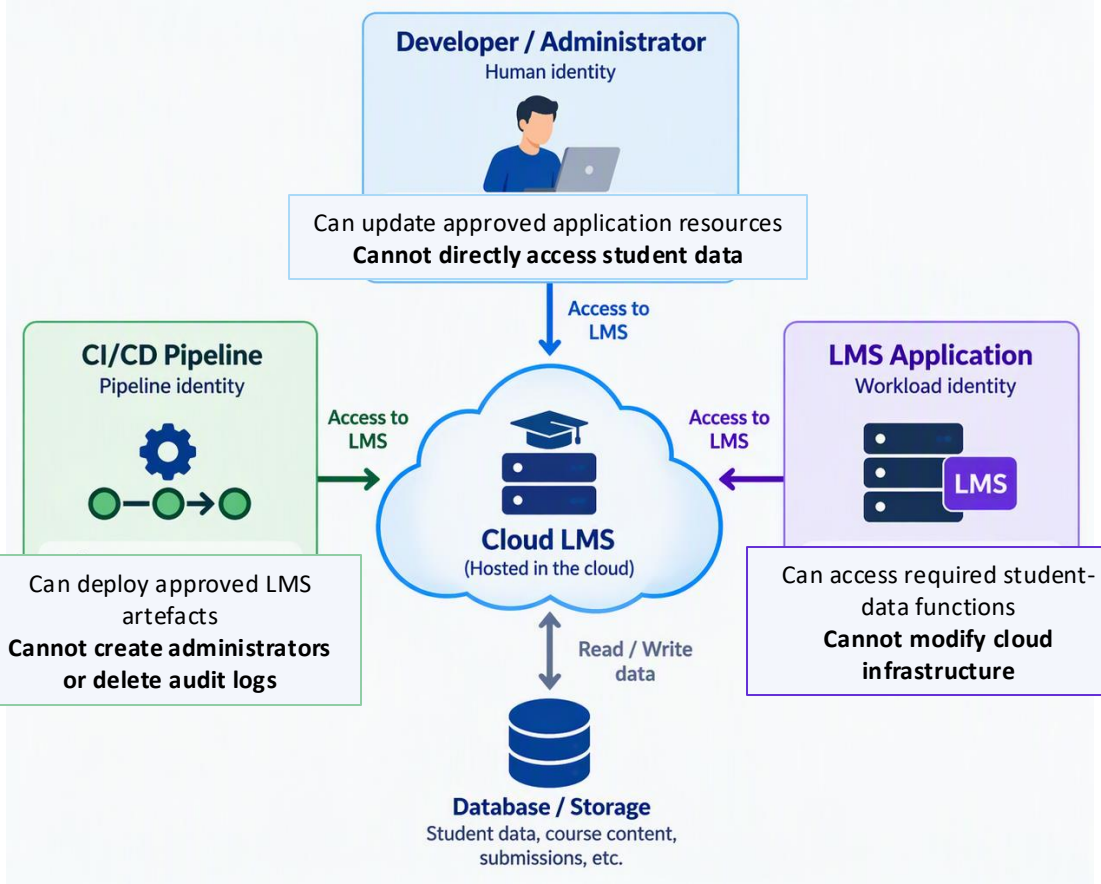
Identity connects the cloud environment

- Every cloud action has an identity.
- Human identity
  - Students, staff, developers and administrators
- Pipeline identity
  - Builds and deploys the LMS
- Workload identity
  - Allows applications to access cloud services
- The important question is not only ***"Where did the request come from?"***, but ***"Who or what made it?"***



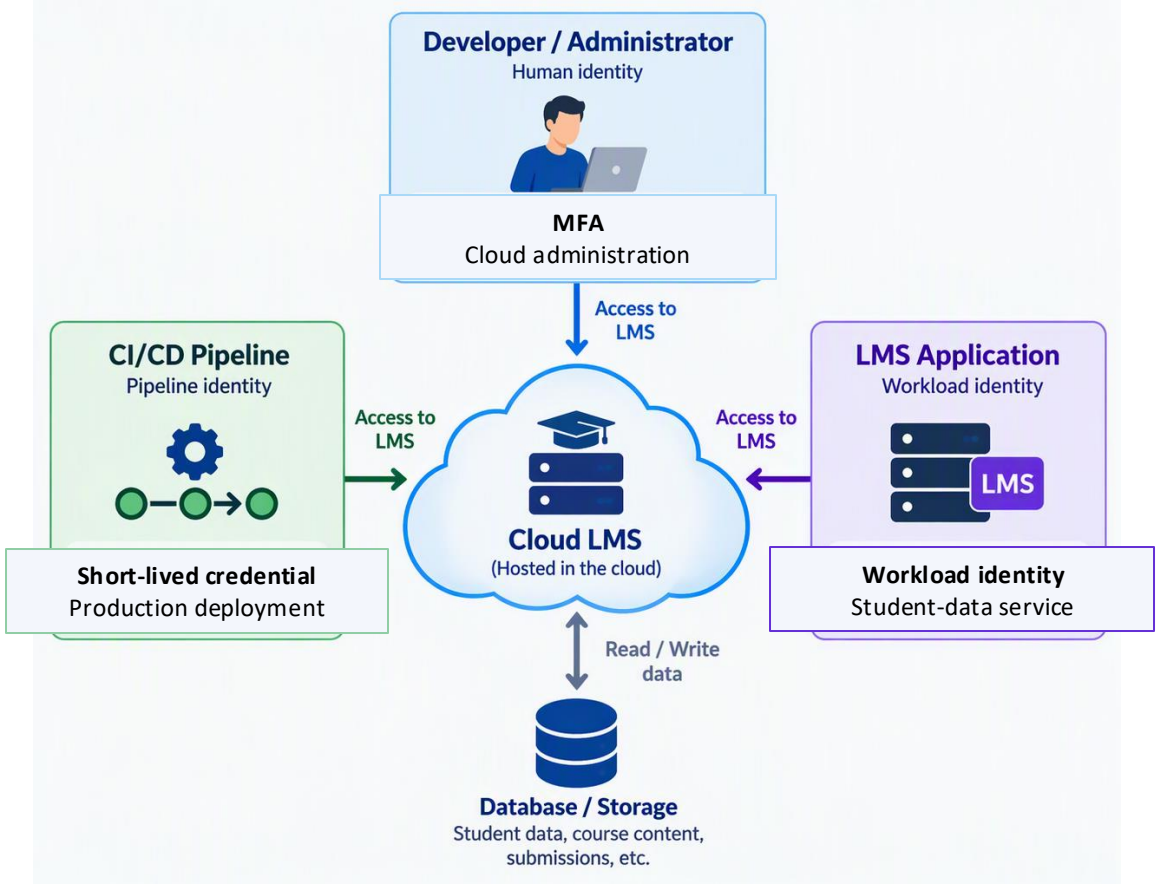
Give every identity only what it needs

- Limit what each identity can do.
- Default deny
  - access must be explicitly granted
- Least privilege
  - grant only required permissions
- Temporary access
  - avoid unnecessary long-lived privileges
- Separation of duties
  - avoid concentrating critical control
- **A compromised identity should not compromise the entire cloud environment.**



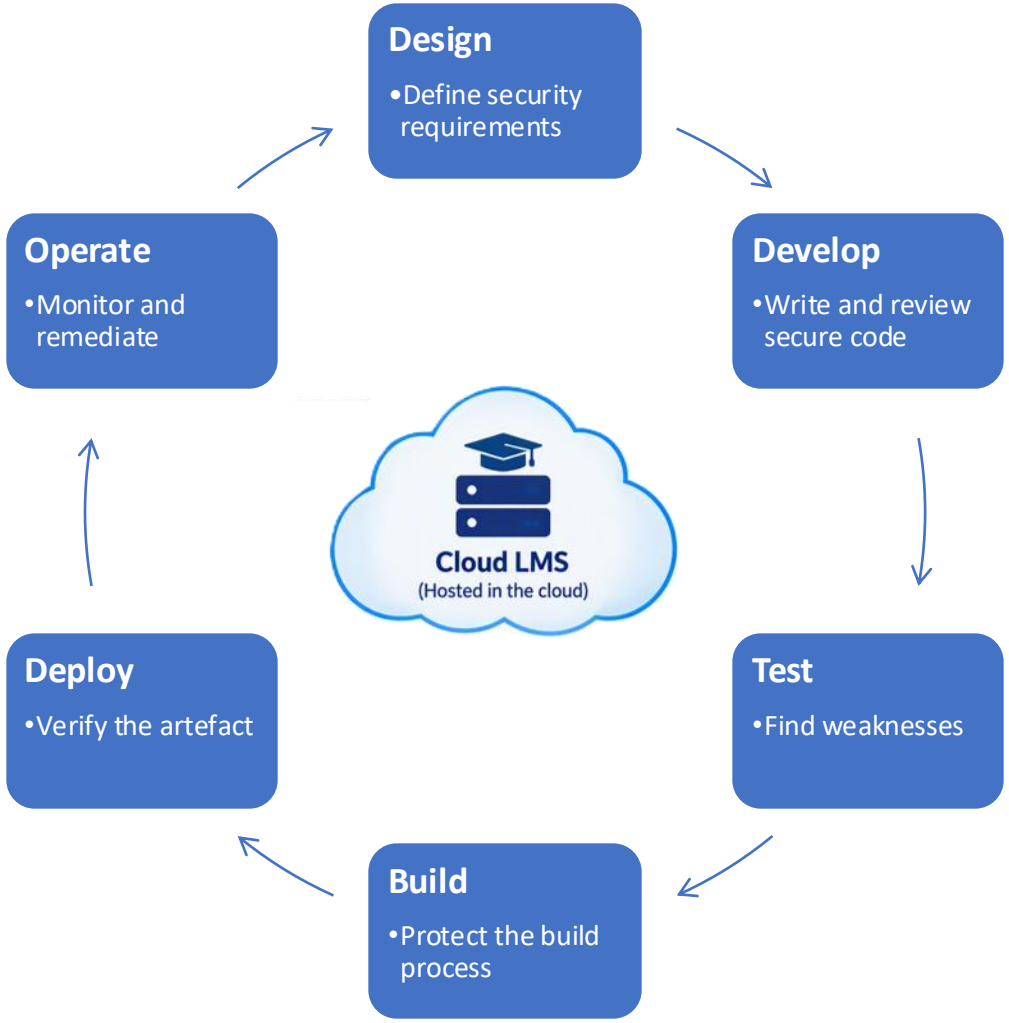
Protect high-risk cloud access

- Require **MFA** for sensitive human access
- Use **short-lived credentials** for workloads and pipelines
- Avoid shared accounts and embedded credentials
- Record privileged and deployment activity
- Revoke access quickly when compromise is suspected
- **Strong permissions are not enough if the credential itself is stolen.**



Secure software is a lifecycle

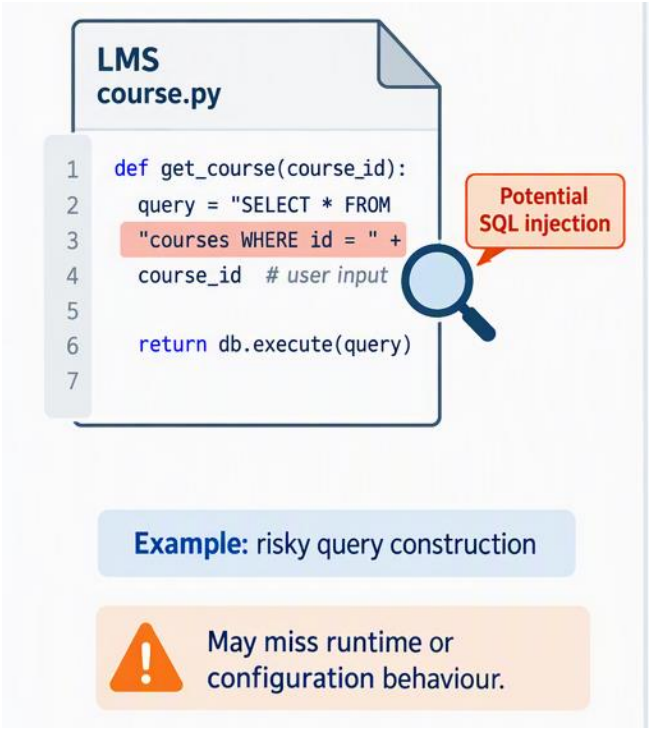
- Security must follow the software lifecycle.
- At every stage:
  - Identify security requirements
  - Protect code and configuration
  - Test before release
  - Verify what is deployed
  - Respond to discovered vulnerabilities
- **Security added only before deployment is already too late.**



Three tests, three different views

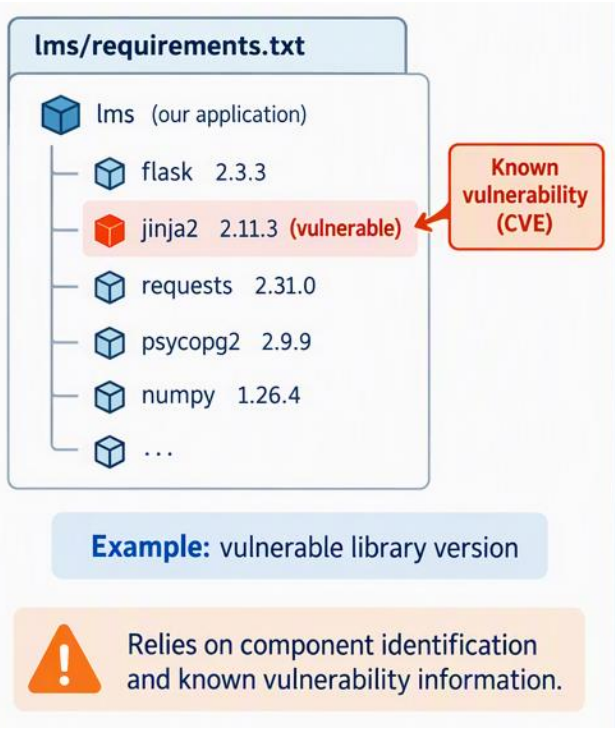
SAST

Analyse our code



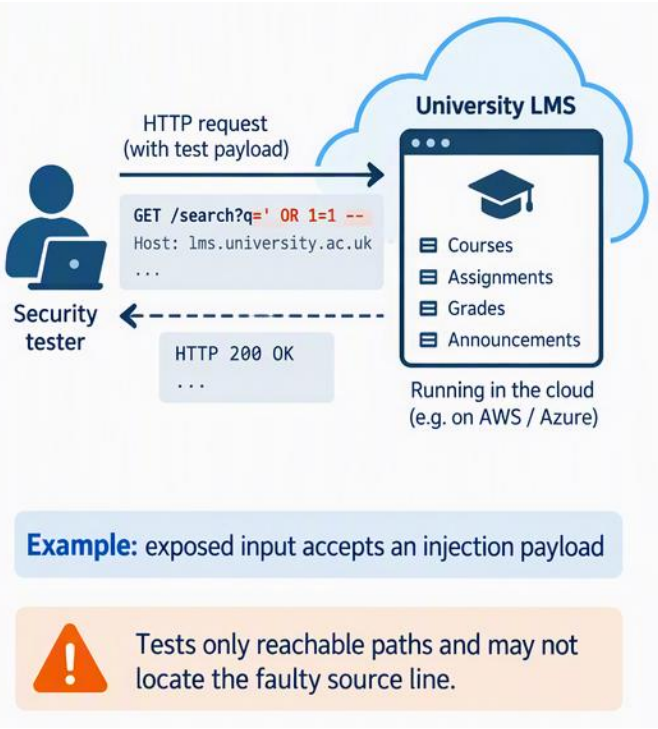
SCA

Analyse dependencies



DAST

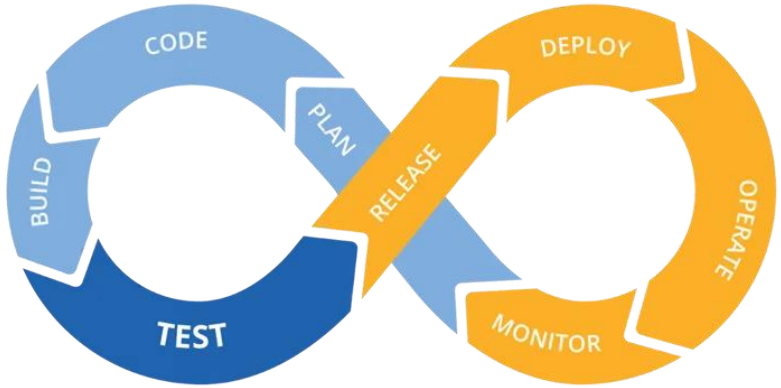
Test the running LMS



They are complementary, no single method provides complete coverage.

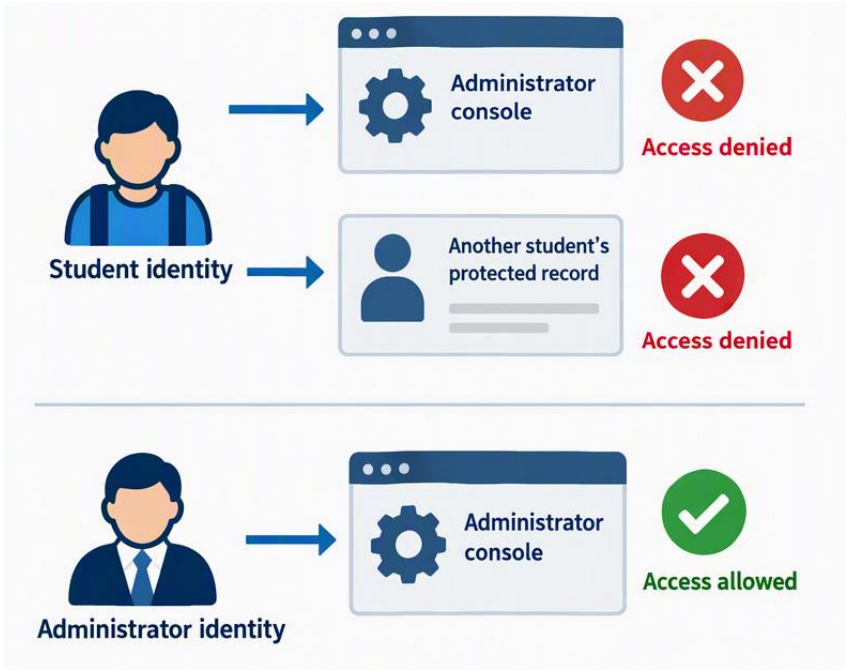
Review the change, not just the code

- Review the complete change.
- Before the LMS reaches production, review:
  - Application code — What behaviour changed?
  - Dependencies — What new components were introduced?
  - Cloud configuration — What access or exposure changed?
  - Pipeline definition — How will the change be built and deployed?
  - Approvals — Who verified the change?
- **A small code change can create a large production impact.**



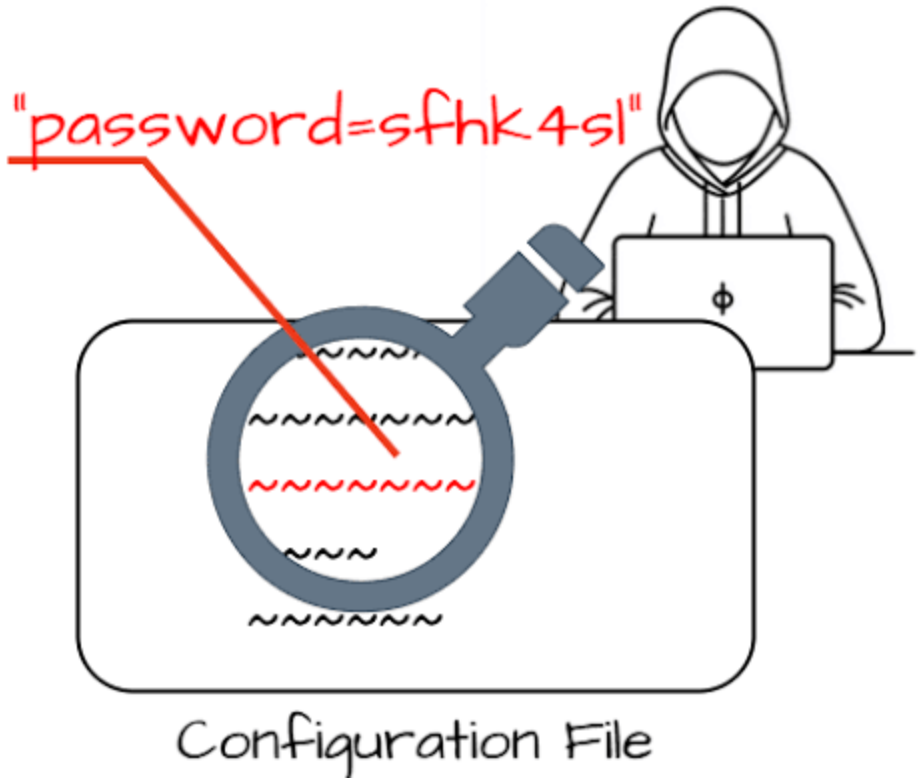
Test security outcomes

- Test what should succeed, and what must fail.
- Permitted action
  - A student can access their own courses and results
- Forbidden role
  - A student cannot open administrator functions
- Forbidden record
  - A student cannot access another student’s protected information
- Invalid input
  - Unexpected data is rejected safely
- **A control is effective only when authorised actions succeed and unauthorised actions fail.**



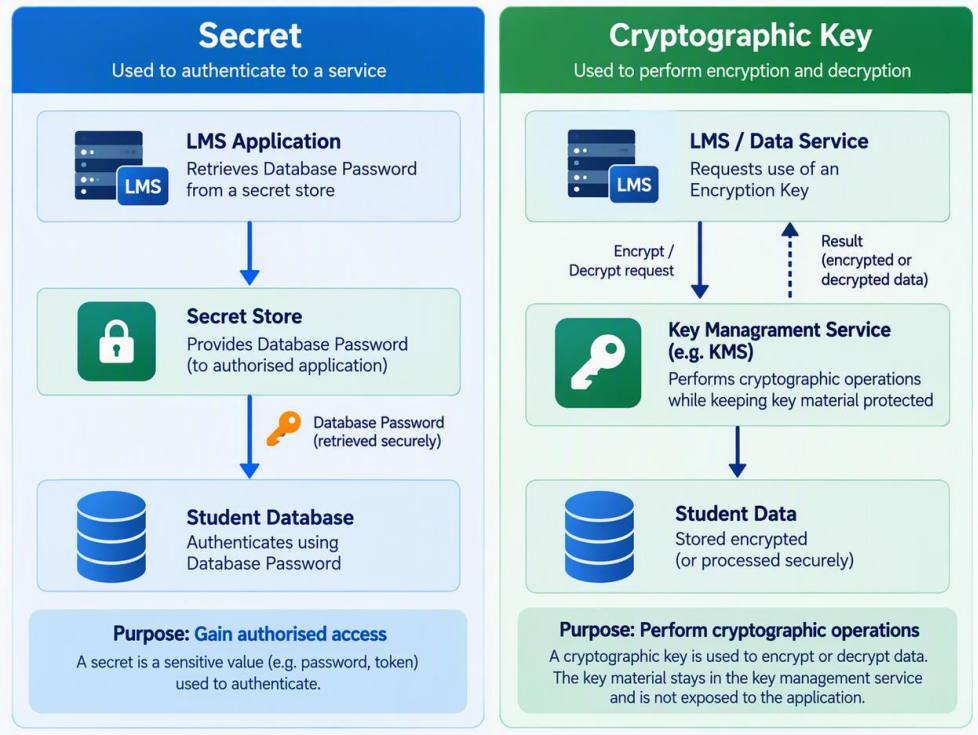
Secrets do not belong in source code

- A committed password is no longer secret.
- Unsafe pattern
  - Database password stored in LMS source or configuration
- Exposure
  - Repository readers and copied artefacts may reveal it
- Safer pattern
  - LMS retrieves the password from a managed secret store at runtime
- After exposure
  - Revoke or rotate the password, deleting the line is not enough
- **Store the reference in code, not the secret.**



Secrets and cryptographic keys are different

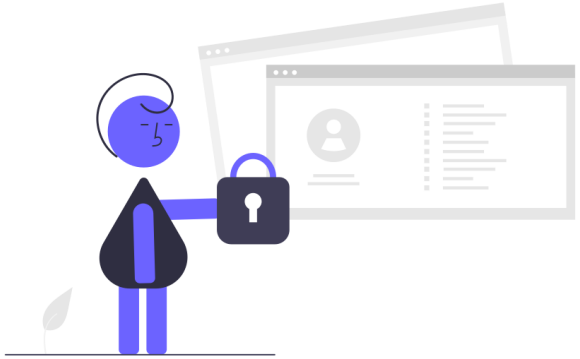
- Secrets and keys protect different things.
- Secret
  - Proves identity or grants access
    Example: LMS database password
- Cryptographic key
  - Performs a cryptographic operation
    Example: Key used to encrypt student data
- Both require
  - Protected storage, controlled access and lifecycle management
- **A password opens the door.**
  A cryptographic key protects the data.



Encryption does not fix poor access control

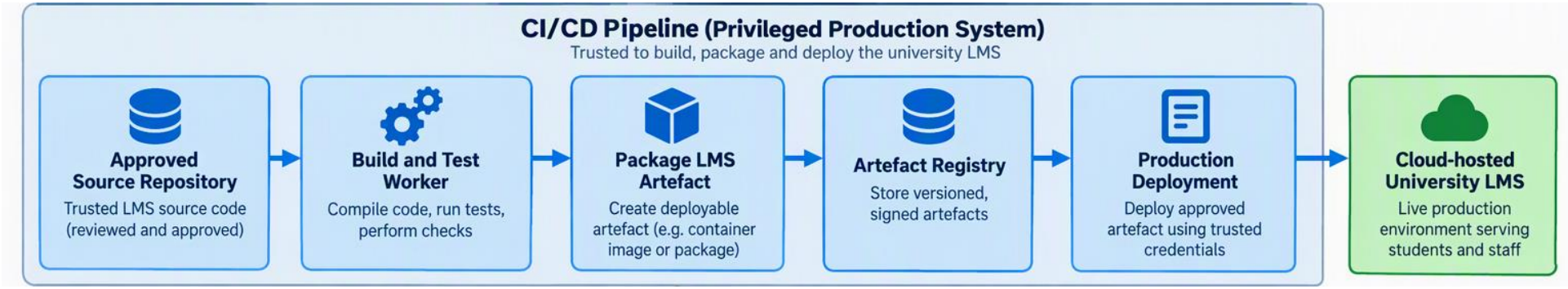
- Encryption protects data, access control decides who can read it.
- At rest
  - Protect stored student records
- In transit
  - TLS protects data moving between systems
- Access control
  - Determines which identities receive usable data
- Overprivileged identity
  - The LMS can still read all records if granted excessive access
- **Encrypt the data. Restrict who can decrypt and use it.**





The pipeline is a privileged production system

- The pipeline is a privileged path to production that can access:
  - Approved source code
  - Build and test environments
  - Artefact repositories
  - Secrets and deployment credentials
  - The production LMS environment
- **If the pipeline is compromised, its output may still appear trusted.**



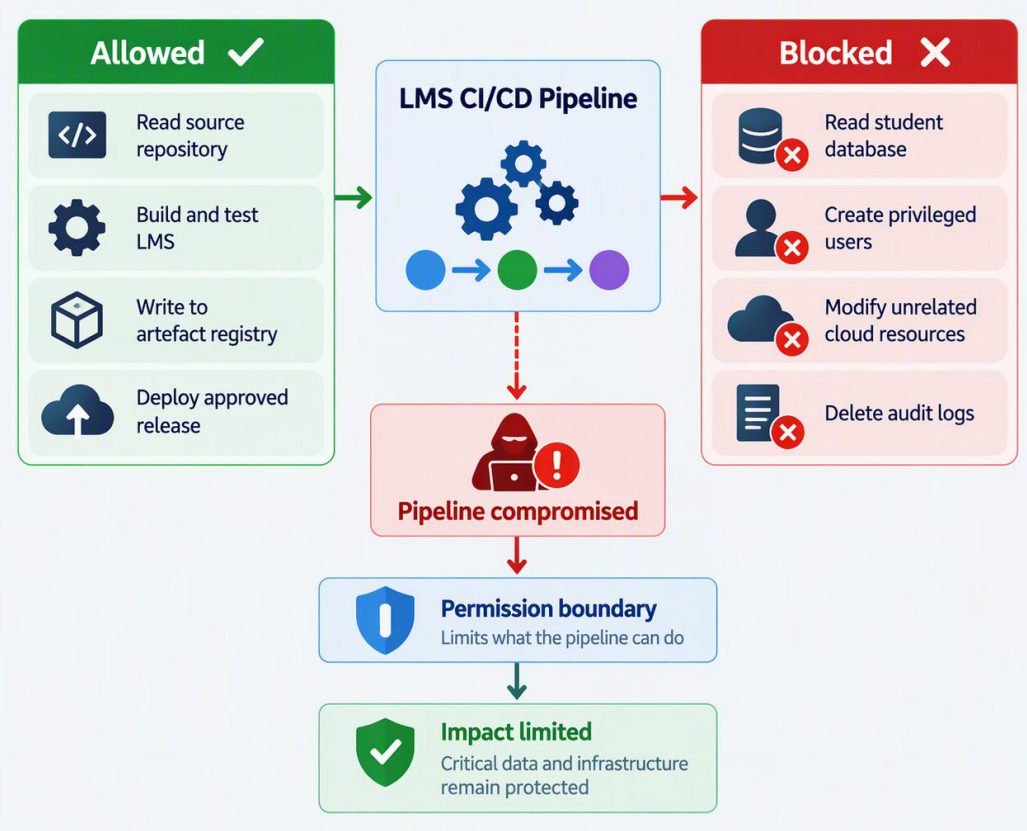
Protect source, branches and build workers

- Protect the path into the pipeline.
  - **Source repository** — restrict who can change code
  - **Protected branches** — require review before merging
  - **Build workers** — isolate trusted build environments
  - **Third-party components** — control what the pipeline can execute
  - **Changes** — record who approved and introduced them
- **If the build path is untrusted, the resulting artefact cannot be trusted automatically.**



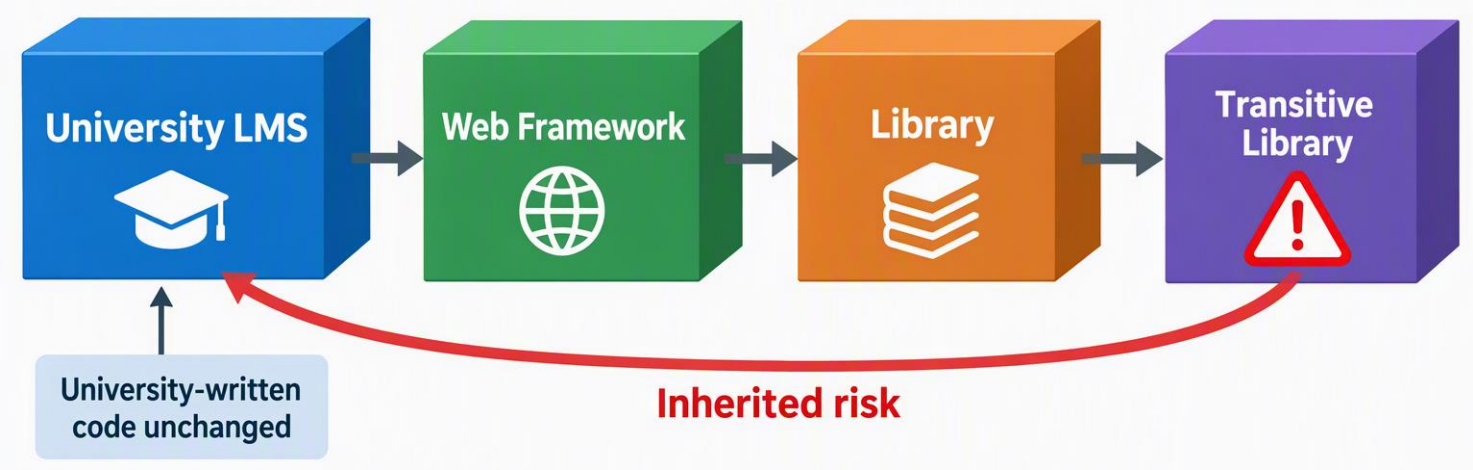
Limit pipeline permissions

- The pipeline should only have the access it needs.
- Required
  - Read approved source
  - Write approved artefacts
  - Deploy the LMS
- Not required
  - Read student records
  - Create administrator accounts
  - Delete security logs
- **A compromised pipeline should not become a cloud administrator.**



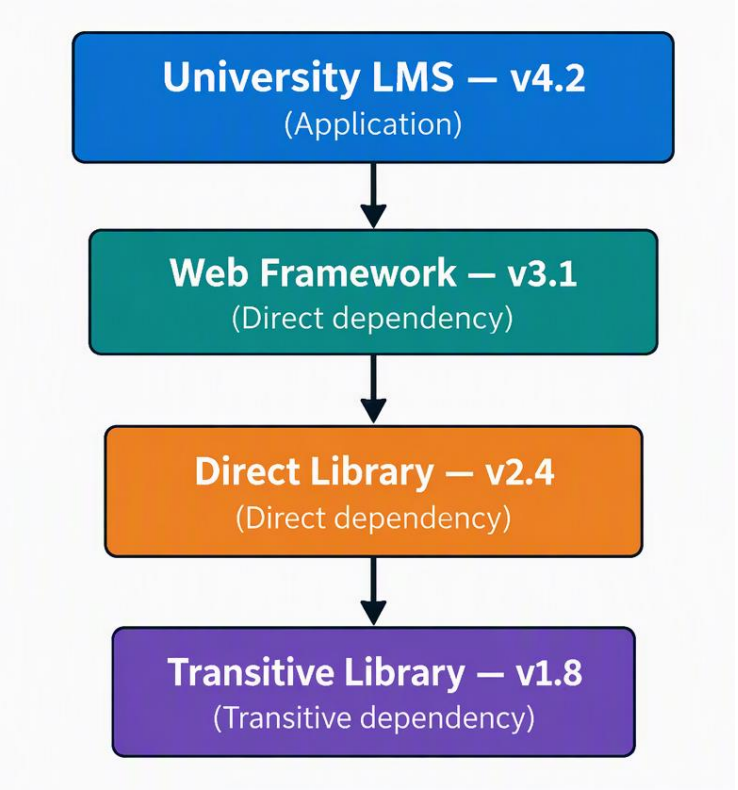
The software supply chain beyond our code

- The LMS includes software we did not write.
- Each component may introduce further dependencies.
- Risk can enter through any link in the chain.
- University-written code may remain unchanged.
- The complete application inherits the component's behaviour.
- **We inherit both capability and risk from our dependencies.**



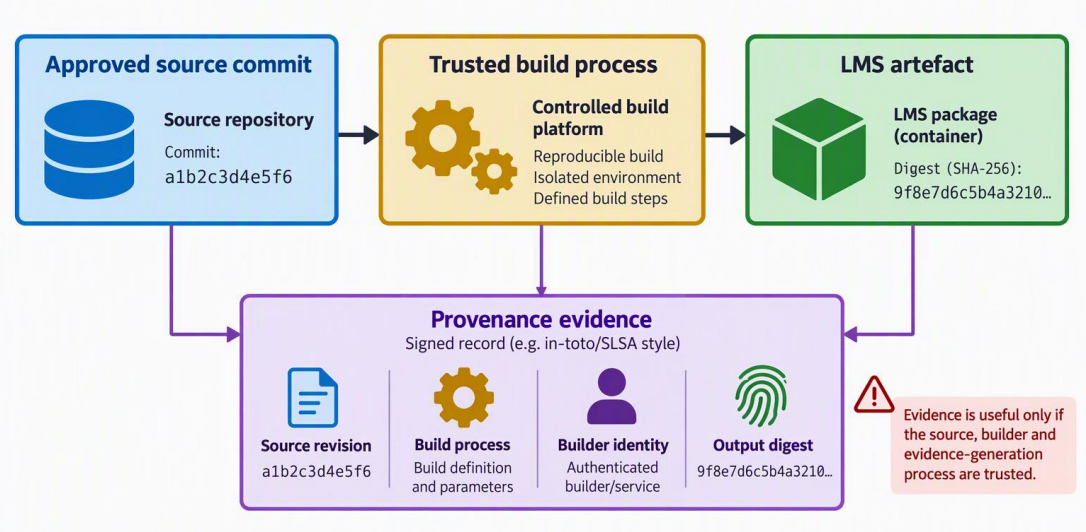
SBOM: What is inside?

- Software Bill of Materials (SBOM): An inventory of the LMS
- A SBOM records:
  - Components included in the release
  - Versions of those components
  - Relationships between dependencies
  - Direct and transitive dependencies
- **An SBOM tells us what is inside, not whether it can be trusted.**



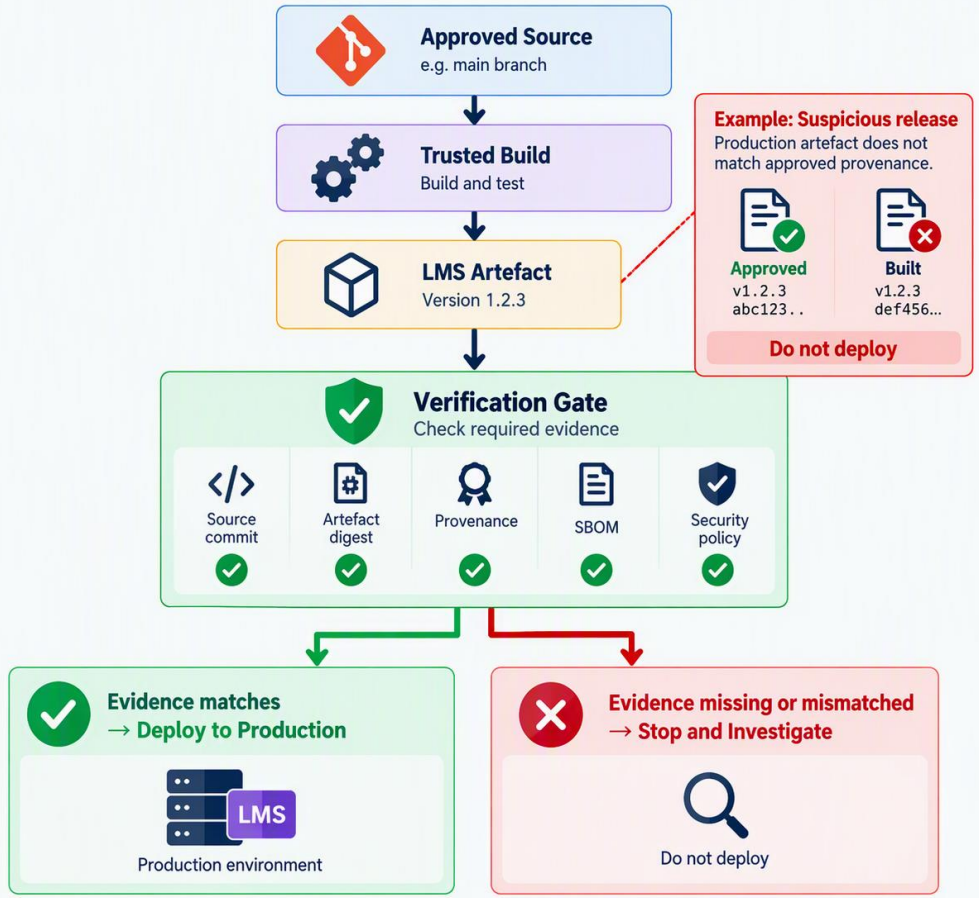
Provenance: Where did the it come from?

- Provenance links the artefact to its origin.
- Provenance records evidence such as:
  - Source revision
  - Build process
  - Builder identity
  - Artefact digest
- **SBOM: What is inside?**
- **Provenance: Where and how was it produced?**



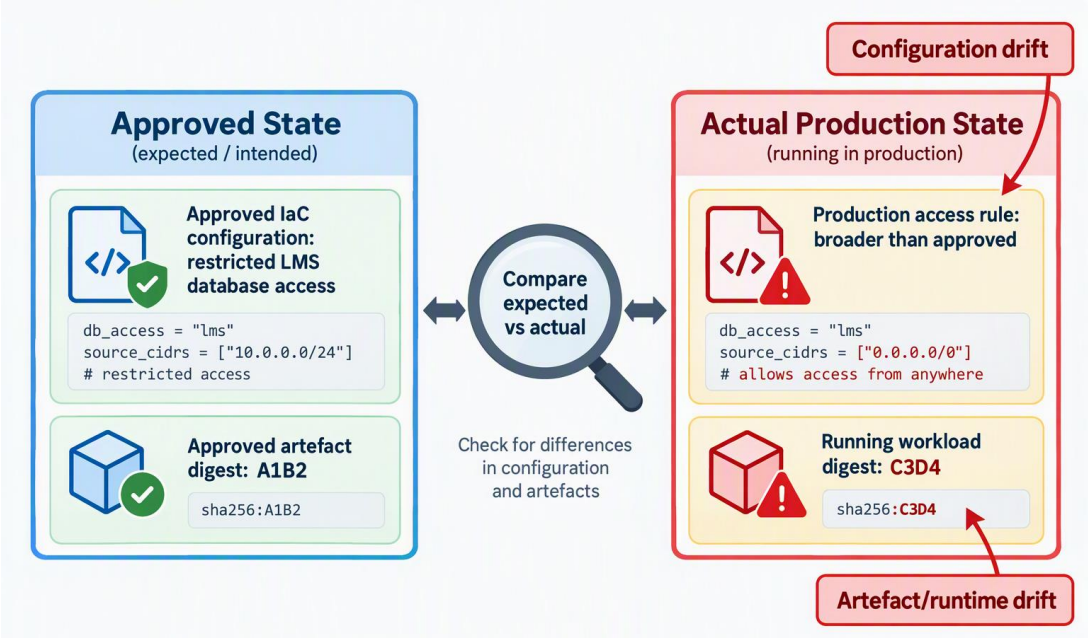
Verify before deployment

- Do not deploy on trust alone.
- Before production, verify:
  - Source: Is it the approved commit?
  - Build: Was it produced by the trusted pipeline?
  - Artefact: Does its identity match the evidence?
  - Evidence: Are SBOM and provenance available?
  - Policy: Does the release meet deployment requirements?
- **If the evidence does not match, stop the deployment.**



Detect configuration and runtime drift

- Does production still match what we approved?
- Approved state
  - Reviewed IaC configuration and verified LMS artefact
- Configuration drift
  - Production settings differ from the approved configuration
- Artefact drift
  - The running LMS differs from the approved artefact
- **Compare expected state with actual state, investigate every unexplained mismatch.**



Preserve the evidence needed to explain

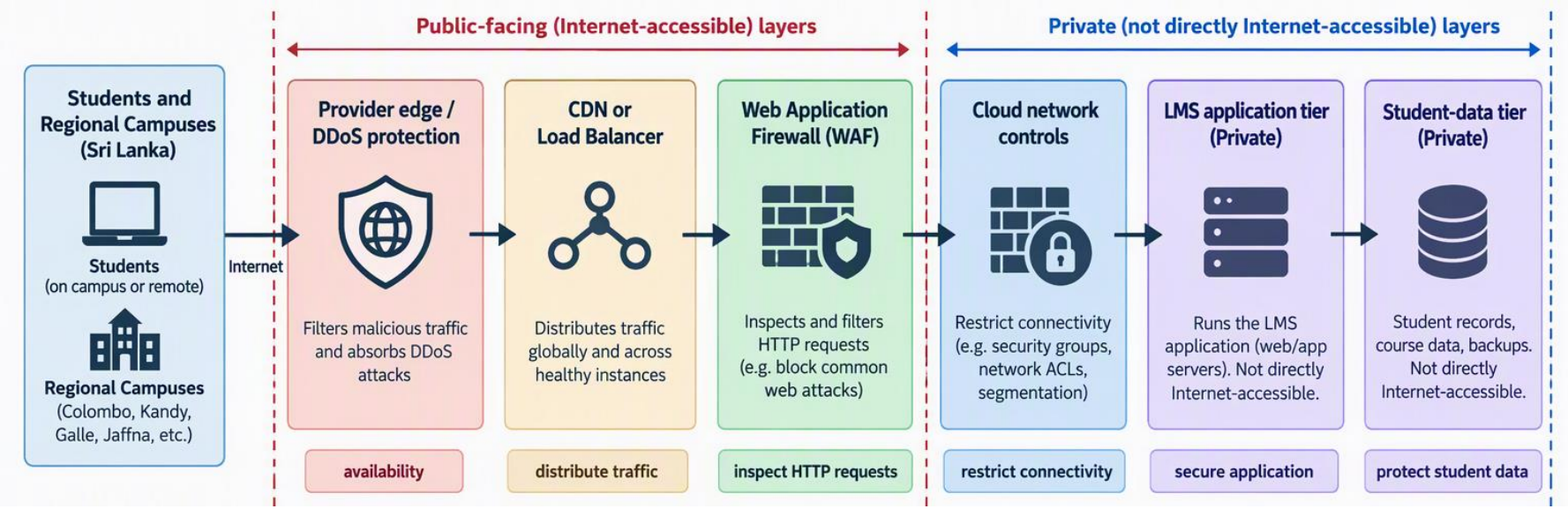
- What evidence would explain the LMS incident?
  - Identity activity:
    - Who authenticated or gained privilege?
  - Cloud changes:
    - What resource or configuration changed?
  - Source and pipeline:
    - Who approved, built and deployed the release?
  - Artefact evidence:
    - Which build entered production?
  - Workload activity:
    - What did the running LMS do?
  - Data access:
    - Which student records were accessed?
- **Evidence should reconstruct who did what, to which resource, and when.**



Layered protection for the public LMS

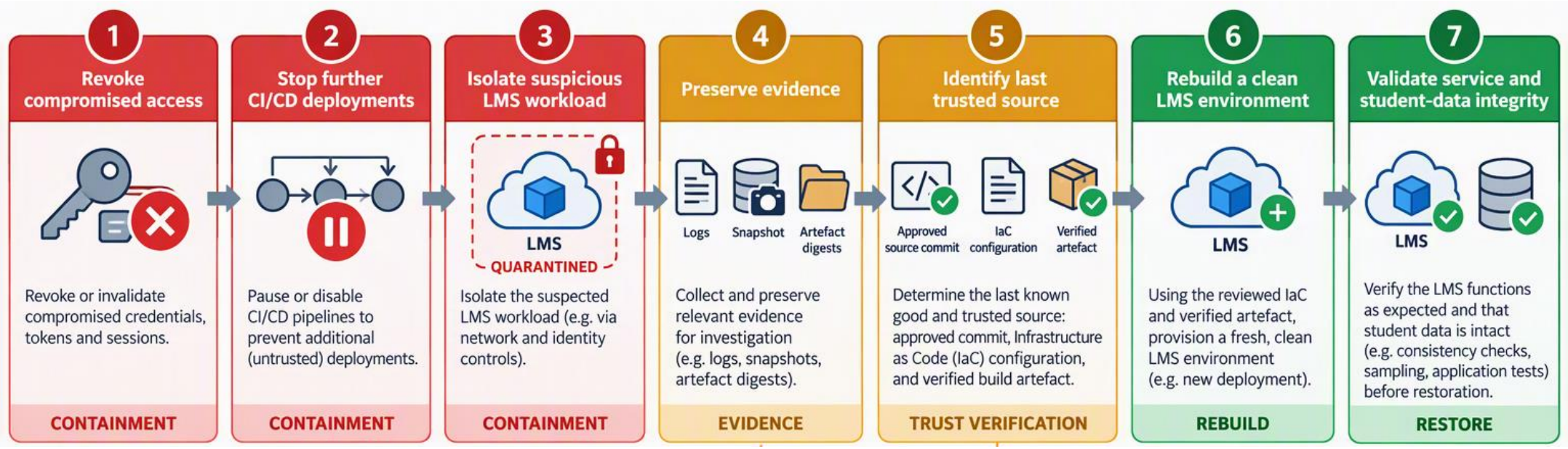
- Protect the public LMS in layers.
  - Distribute or absorb traffic
  - Inspect web requests
  - Restrict connectivity
  - Keep student data non-public

- **Each layer reduces risk. No single layer is sufficient.**



Contain and rebuild the LMS

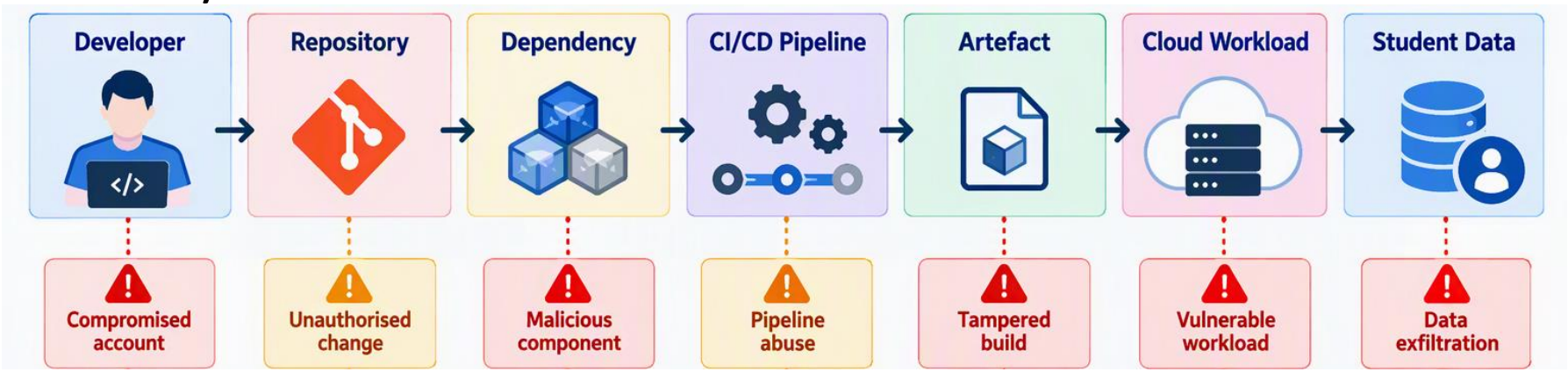
- Contain first. Restore from trusted state.



- **Do not restore availability by reintroducing uncertainty.**

Final challenge: Break the attack chain

- Where would you break the attack chain?



- For each stage, identify:
  - **Prevent**: What could stop the attack?
  - **Detect**: What would reveal it?
  - **Contain**: How could we limit the impact?
  - **Recover**: What trusted state would we restore?
- **Cloud security depends on protecting the complete chain of trust.**