Capstone Project Categories

Capstone Project Categories

This document is your primary reference for choosing and developing a capstone project. Each category represents a *type of system* that naturally surfaces specific learning tensions.

How to use this document:

1. Read the categories to find one that interests you
2. Use the **early-resistance start prompt** to begin work in Week 1
3. Avoid the **shallow failure** pattern — it's how students accidentally avoid learning
4. Reference the **cloud manifestation** for concrete AWS services
5. Use the **CCT example** as a template for your own claims

The CCT pattern: state what you claim (**Clause**), describe how you enforce it (**Control**), then specify how someone could verify it (**Test**). If you cannot write a test, you do not yet understand your claim.

---

1. Coordination Under Partial Trust

Essence: Multiple actors must cooperate without full mutual trust.

Naturally stresses:

- Commitment vs Flexibility
- Narrative Coherence vs Epistemic Honesty

Cloud manifestation:

- Cross-account IAM roles and assume-role chains
- Multi-region data with jurisdiction constraints
- Federated identity (SAML, OIDC) across organizational boundaries
- Service-to-service authentication between managed services

Shallow failure: Treating trust as binary ("admin vs user"). Real systems have graduated trust, contested authority, and legitimate disagreement.

What strong work shows:

- Explicit disagreement between actors
- Audit trails or reconciliation logic that reveal tradeoffs

Early-resistance start prompt:

> Design a system where *two parties disagree* about a shared state update, and neither is allowed to be a superuser. Implement the disagreement first; resolution comes later.

CCT Example:

Clause	Control	Test
"Cross-account writes are auditable"	IAM role + CloudTrail logging in both accounts	Query audit logs; verify every cross-account write has entries in both source and destination accounts

---

2. Systems That Remember (and Forget)

Essence: Deciding what to retain, summarize, or delete over time.

Naturally stresses:

- Depth vs Breadth
- Optimization vs Understanding

Cloud manifestation:

- S3 lifecycle policies and intelligent tiering
- DynamoDB TTL for automatic expiration
- Glacier/Deep Archive for cold storage economics
- GDPR right-to-be-forgotten across distributed services
- CloudWatch log retention policies

Shallow failure: "Store everything forever" designs that assume storage is free. Real systems must delete, and deletion has consequences.

What strong work shows:

- A forced deletion or summarization decision
- Reflection on what information loss implies

Early-resistance start prompt:

> Impose a hard storage budget. When it is exceeded, the system must delete *something*. Decide what and justify why.

CCT Example:

Clause	Control	Test
"User data is deleted within 30 days of account closure"	DynamoDB TTL + S3 lifecycle + scheduled Lambda for derived data	Close test account; query all storage locations after 30 days; no user data should remain

3. Failure-First Systems

Essence: The system exists because failure is expected.

Naturally stresses:

- Control vs Realism
- Commitment vs Flexibility

Cloud manifestation:

- Multi-AZ deployments and AZ failure scenarios
- Spot instance termination handling
- Lambda cold starts and timeout behavior
- RDS failover and replica promotion
- S3 eventual consistency (for overwrites/deletes)

Shallow failure: Simulated failures with no consequences. If your "failure injection" doesn't change system behavior or corrupt state, you haven't tested failure.

What strong work shows:

- A failure that corrupts state or breaks assumptions
- Evidence that fixes required redesign, not patches

Early-resistance start prompt:

> Introduce a failure mode that corrupts partial state. Your first version must *exhibit* the bug before you attempt to fix it.

CCT Example:

Clause	Control	Test
"Service survives single-AZ failure"	Multi-AZ deployment + ALB health checks + auto-scaling	Terminate all instances in one AZ; verify requests continue to succeed within SLA

---

4. Governance and Constraint Engines

Essence: The system's job is enforcement, not computation.

Naturally stresses:

- Narrative Coherence vs Epistemic Honesty
- Optimization vs Understanding

Cloud manifestation:

- AWS Organizations Service Control Policies (SCPs)
- AWS Config rules and conformance packs
- GuardDuty and Security Hub findings
- Budget alerts and spending limits
- Tag enforcement policies

Shallow failure: Static rules that never evolve and have no override path. Real governance requires exceptions, and exceptions require accountability.

What strong work shows:

- Explicit override paths
- Clear explanation of who bears responsibility when rules conflict

Early-resistance start prompt:

> Implement a policy that must sometimes be overridden. Record *who*, *why*, and *with what consequence*.

CCT Example:

Clause	Control	Test
"Production resources cannot be deleted without approval"	SCP denying `Delete*` actions + Step Functions approval workflow for exceptions	Attempt direct deletion as admin; must fail. Submit deletion request; must require human approval

---

5. Invisible Infrastructure

Essence: Supporting systems that other systems depend on.

Naturally stresses:

- Depth vs Breadth
- Control vs Realism

Cloud manifestation:

- VPC networking, NAT gateways, and transit gateways
- Route 53 DNS resolution and failover
- Managed service dependencies (RDS, ElastiCache, SQS)
- Secrets Manager and Parameter Store
- Certificate Manager and key rotation

Shallow failure: Single-consumer or toy deployments. Infrastructure only reveals its nature when multiple dependents fail differently.

What strong work shows:

- Multiple dependents behaving differently under stress
- Reflection on blast radius

Early-resistance start prompt:

> Create two services that depend on your infrastructure. Break the infrastructure and observe how each fails differently.

CCT Example:

Clause	Control	Test
"DNS resolution completes within 100ms for internal services"	Route 53 private hosted zones + health checks	Measure resolution latency under load; verify p99 < 100ms. Simulate unhealthy endpoint; verify failover completes within TTL

---

6. Economics-Shaped Systems

Essence: Cost is a first-class constraint that shapes behavior.

Naturally stresses:

- Optimization vs Understanding
- Commitment vs Flexibility

Cloud manifestation:

- Reserved vs on-demand vs spot instance pricing
- Data transfer and egress costs
- Tiered API pricing (requests, storage classes)
- Cost allocation tags and chargeback models
- Savings Plans and commitment discounts

Shallow failure: "Costs would be low" with no model. If you haven't measured or projected costs, you don't understand your system's economics.

What strong work shows:

- A concrete budget constraint
- Functionality that degrades under cost pressure

Early-resistance start prompt:

> Give your system a weekly budget. When the budget is exceeded, functionality must degrade gracefully.

CCT Example:

Clause	Control	Test
"Monthly spend stays under $50"	Budget alarm at $40 + Lambda that scales down non-essential services at $45 + hard stop at $50	Generate synthetic load that would exceed budget; verify degradation triggers; verify hard stop prevents overage

---

7. Edge-Core Hybrid Systems

Essence: Computation split across constrained edge and central core.

Naturally stresses:

- Control vs Realism
- Depth vs Breadth

Cloud manifestation:

- Lambda@Edge and CloudFront Functions
- IoT Core and Greengrass for device fleets
- Outposts and Local Zones
- S3 Transfer Acceleration and regional caching
- Wavelength for ultra-low latency edge

Shallow failure: Treating the edge as just another client with no behavior change during disconnects. The edge is interesting precisely because it must operate autonomously.

What strong work shows:

- Behavior changes during disconnection
- Explicit placement rationale

Early-resistance start prompt:

> Simulate a multi-hour network partition. Decide what *must* still work and what is allowed to fail.

CCT Example:

Clause	Control	Test
"Edge devices continue operating during core disconnection"	Greengrass local shadow + store-and-forward queue	Disconnect network from edge device; verify local operations continue; reconnect; verify queued data syncs to core

8. Human-in-the-Loop Pipelines

Essence: Automation is deliberately incomplete.

Naturally stresses:

- Narrative Coherence vs Epistemic Honesty
- Control vs Realism

Cloud manifestation:

- Step Functions with manual approval tasks
- SES/SNS for human notification loops
- SageMaker Ground Truth for labeling workflows
- Cognito for identity in approval chains
- EventBridge for timeout and escalation

Shallow failure: Treating humans as perfect oracles with instant response. Real humans are slow, make mistakes, and sometimes don't respond at all.

What strong work shows:

- Queueing behavior
- Clear ownership of decisions

Early-resistance start prompt:

> Insert a human decision step with unpredictable latency. Show how the system behaves while waiting.

CCT Example:

Clause	Control	Test
"High-value transactions require human approval within 4 hours"	Step Functions wait state + SNS notification + CloudWatch alarm for SLA breach	Submit high-value transaction; verify blocked pending approval. Let timeout expire; verify escalation triggers and SLA breach is logged

Note: examples like this can *also* be subject to external audit (e.g., an outside firm audits the implementation to verify that it meets the SLA.) This can be enforced via a test, such as one in which an **attested verification of audit** is required - no attested audit confirmation available? The test *fails*. This demonstrates how even external events can be internally enforced.

9. Time-Dominated Systems

Essence: Time itself is the core difficulty.

Naturally stresses:

- Control vs Realism
- Optimization vs Understanding

Cloud manifestation:

- EventBridge scheduled rules and cron expressions
- DynamoDB TTL and time-based expiration
- CloudWatch Logs Insights time-range queries
- Step Functions timeouts and heartbeats
- SQS visibility timeout and delay queues

Shallow failure: Assuming perfect clocks. Distributed systems have clock skew, and time-based logic fails in surprising ways when clocks disagree.

What strong work shows:

- A correctness failure caused solely by time
- Explicit mitigation strategies

Early-resistance start prompt:

> Intentionally skew clocks between components. Demonstrate a correctness failure caused *only* by time.

CCT Example:

Clause	Control	Test
"Session tokens expire after 15 minutes"	DynamoDB TTL on session records + Lambda authorizer checking expiry	Create session; verify access works. Wait 15 minutes; verify access denied. Query DynamoDB; verify record deleted by TTL

---

10. Ethically Load-Bearing Systems

Essence: Failures have moral consequences, not just technical ones.

Naturally stresses:

- Narrative Coherence vs Epistemic Honesty
- Commitment vs Flexibility

Cloud manifestation:

- Data residency and region selection (jurisdiction)
- KMS key management and custody (who holds keys?)
- CloudTrail audit logs (accountability and transparency)
- Consent management and GDPR compliance
- Split-key architectures (resisting compulsion)

Shallow failure: Ethical claims with no architectural impact. If your ethics don't constrain your design, they're not load-bearing.

What strong work shows:

- Design choices that privilege one value at a clear cost
- Explicit acknowledgment of unresolved risk

Early-resistance start prompt:

> Define a failure mode where correctness conflicts with harm reduction. Show how the system chooses.

CCT Example:

Clause	Control	Test
"User data never leaves EU jurisdiction"	S3 bucket in eu-west-1 + bucket policy denying cross-region replication + SCP blocking non-EU resource creation	Attempt to create S3 replication to us-east-1; must fail. Attempt to read data from US-based service; must fail. Query CloudTrail; verify no data access from non-EU regions

Note: this example is one where the simple test *can fail*. The United States asserts the right to obtain *any* data that can be accessed by a US company, thus Amazon could be forced to surrender this data anyway (see the **CLOUD Act (https://en.wikipedia.org/wiki/CLOUD_Act)** of 2018).

Choosing Your Category

Pick a category that will *force you* to confront what you do not yet understand. The goal is not success — it is **resistance**.

If everything in your project "just works," you have not pushed far enough. The learning lives in the friction.

See also:

- **Capstone Assessment Rubric** — how your work will be evaluated
- **Wild Project Seeds** — provocative ideas to expand your imagination

- **Worked Seedling: Split-Key Envelope** — one example developed in detail