*This table maps the techniques of the taxonomy to types of techniques (see discussion section)*

| **Refined category** | **Technique** | **Techinique Type** | |
|---|---|---|---|
| (C1.1) Integration With Clients | | | |
| | 1.1.1 API facade | Architecture | |
| | 1.1.2 Edge server facade | Architecture | |
| | 1.1.3 API gateway facade | Architecture | |
| | 1.1.4 API facade per client type | Architecture | |
| | 1.1.5 Independent choice of communication technology | Architecture | |
| (C1.2) Integration of 3rd-party Systems Into the Application | | | |
| | 1.2.1 Proxy microservice | Architecture | |
| | 1.2.2 Data replication proxy | Architecture | |
| | 1.2.3 CQRS proxy | Architecture | |
| | 1.2.4 Gradually replace the legacy system | Architecture | |
| | 1.2.5 Treat legacy system like a microservice | Architecture | |
| | 1.2.6 ESB to decouple from legacy system | Architecture | |
| (C1.3) Integration Into an Application Landscape | | | |
| | 1.3.1 Service/API registry | Process | Tool to support coordination of org. units: make services/APIs easier to discover |
| | 1.3.2 Document microservice metadata | Process | Tool to support coordination of org. units: make services/APIs easier to understand/discover |
| | 1.3.3 Enterprise-wide standardization | Process | Tool to support coordination of org. units: reduce mental load by standardization |
| | 1.3.4 Enterprise service wrapper | Architecture | |
| (C3.1) Conceptual Integration | | | |
| | Service cut: | | |
| | 3.1.1 Evaluate cut with proof of concepts | Architecture | Evaluation of arch. decisions |
| | 3.1.2 Avoid LoC metric for evaluation | Architecture | Evaluation of arch. decisions |
| | 3.1.3 Decentralize the service cut | Organization | Responsibility of different org. units |
| | 3.1.4 Cut by non-functional characteristics | Architecture | |
| | 3.1.5 Cut by functional proximity | Architecture | |
| | 3.1.6 Cut by Domain-Driven Design | Architecture | |
| | 3.1.7 Cut by data entities and consistency needs | Architecture | |
| | 3.1.8 Cut by use-case | Architecture | |
| | 3.1.9 Cut by data-flow | Architecture | |
| | Dataflows: | | Type of data exchange and transactions among microservices part of architectural view |
| | 3.1.10 Question transactions on domain level | Architecture | |
| | 3.1.11 Avoid transactions over multiple microservices | Architecture | |
| | 3.1.12 Data replication | Architecture | |
| | Workflows: | | Workflows between microservices part of architectural view |
| | 3.1.13 Choreography over orchestration | Architecture | |
| | 3.1.14 Align synchronicity to business flow | Architecture | |
| | Storage management: | | |
| | 3.1.15 Decentralize conceptual models | Architecture | |
| | 3.1.16 Clear responsibilities for parts of the data | Organization | Responsibility of different org. units |
| | Location of business logic: | | |
| | 3.1.17 No domain logic into infrastructure | Architecture | |
| | 3.1.18 No sharing of domain-specific code | Implementation | Code sharing is implementation specific topic |
| | User auth: | | |
| | 3.1.19 Centralized SSO | Architecture | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 3.1.20 Token-based authentication | | Architecture | | | | | | |
| | | 3.1.21 Propagate security context via headers | | Architecture | | | | | | |
| | | 3.1.22 Propagate security context via tokens | | Architecture | | | | | | |
| | | UI integration: | | | | | | | | |
| | | 3.1.23 Only share context information between UIs | | Architecture | | | | | | |
| | | 3.1.24 UI as part of each microservice | | Architecture | | | | | | |
| | | 3.1.25 UI suites | | Architecture | | | | | | |
| | | 3.1.26 Micro-frontends | | Architecture | | | | | | |
| | | Conceptual error handling: | | | Architectural decisions how to cope with errors | | | | | |
| | | 3.1.27 Design for failure | | Architecture | | | | | | |
| | | 3.1.28 Compensations in workflows | | Architecture | | | | | | |
| | | 3.1.29 Degradation of functionality | | Architecture | | | | | | |
| | | 3.1.30 Domain-motivated alternatives | | Architecture | | | | | | |
| (C3.2) Communication Integration | | | | | | | | | | |
| | | General: | | | | | | | | |
| | | 3.2.1 Align technical communication style to the nature of the busines | | Implementation | | | | | | |
| | | Communication security: | | | | | | | | |
| | | 3.2.2 Service-to-service authentication | | Architecture | Implementation detail, but usually part of architectural view | | | | | |
| | | 3.2.3 Encrypt service-to-service communication | | Architecture | Implementation detail, but usually part of architectural view | | | | | |
| | | API contracts: | | | | | | | | |
| | | 3.2.4 Use APIs to decouple from implementation details | | Architecture | | | | | | |
| | | 3.2.5 Resilient consumers | | Implementation | | | | | | |
| | | 3.2.6 Backward-compatible APIs | | Implementation | | | | | | |
| | | 3.2.7 Hypermedia to reduce coupling | | Implementation | | | | | | |
| | | 3.2.8 API versioning | | Implementation | | | | | | |
| | | 3.2.9 Consumer-driven contract testing | | Implementation | | | | | | |
| | | Communication error handling: | | | | | | | | |
| | | 3.2.10 Circuit breaker and fail fast | | Implementation | | | | | | |
| | | 3.2.11 Dead letter queue | | Architecture | Architectural decision about interaction | | | | | |
| | | 3.2.12 Bulkheads | | Implementation | | | | | | |
| | | 3.2.13 Timeouts | | Implementation | | | | | | |
| | | 3.2.14 Bounded retries | | Implementation | | | | | | |
| | | 3.2.15 Domain-motivated implementation details | | Implementation | | | | | | |
| (C3.3) Deployment Integration | | | | | | | | | | |
| | | General: | | | | | | | | |
| | | 3.3.1 CI/CD for automated deployment | | Operation | | | | | | |
| | | 3.3.2 Immutable deployments | | Operation | | | | | | |
| | | 3.3.3 Reduce deployment coordination | | Process | | | | | | |
| | | 3.3.4 Sidecars/service meshes | | Operation | | | | | | |
| | | Service configuration: | | | | | | | | |
| | | 3.3.5 Avoid hardcoded configurations | | Implementation | | | | | | |
| | | 3.3.6 Avoid default values | | Implementation | | | | | | |
| | | 3.3.7 Environment variables for configuration | | Implementation | | | | | | |
| | | 3.3.8 Configuration server for configuration | | Architecture | | | | | | |
| | | 3.3.9 Configuration/deployment as code | | Operation | | | | | | |
| | | 3.3.10 Internal integration proxy to reduce coupling | | Architecture | Architectural element | | | | | |
| | | 3.3.11 DNS for routing | | Operation | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 3.3.12 Service instance discovery | | Architecture | Architectural element | | | | |
| | | 3.3.13 Service instance discovery by message broker | | Architecture | Architectural element | | | | |
| | | Deployment environments: | | | | | | | |
| | | 3.3.14 Virtualize the network | | Operation | | | | | |
| | | 3.3.15 Offer single-node deployment | | Operation | | | | | |
| | | 3.3.16 Provide resources as a service (Cloud) | | Operation | | | | | |
| | | 3.3.17 FaaS/serverless platform to abstract infrastructure | | Operation | | | | | |
| | | 3.3.18 Cluster management by container orchestrator | | Operation | | | | | |
| | | Zero-downtime deployment: | | | | | | | |
| | | 3.3.19 Rollbacks | | Operation | | | | | |
| | | 3.3.20 Rolling updates | | Operation | | | | | |
| | | 3.3.21 Canary releases | | Operation | | | | | |
| | | 3.3.22 Blue-green deployments | | Operation | | | | | |
| | | Deployment artifacts: | | | | | | | |
| | | 3.3.23 Containers as portable deployment artifacts | | Operation | | | | | |
| | | 3.3.24 Artifact registry | | Operation | | | | | |
| (C3.4) Global Knowledge Integration | | | | | | | | | |
| | | Understanding the system: | | | | | | | |
| | | 3.4.1 Standardize location of microservice documentation | | Process | | | | | |
| | | 3.4.2 Responsibility documentation | | Process | | | | | |
| | | 3.4.3 Standardize API documentation | | Process | | | | | |
| | | Organizational structure: | | | | | | | |
| | | 3.4.4 Microservice managed by one team | | Organization | | | | | |
| | | 3.4.5 Align architecture with org structure | | Organization | | | | | |
| | | 3.4.6 Overarching organizational framework | | Process | | | | | |
| | | 3.4.7 Push more responsibility to teams | | Organization | | | | | |
| | | 3.4.8 Group services based on domain proximity | | Organization | | | | | |
| | | Coordination between teams: | | | | | | | |
| | | 3.4.9 Establish a common vocabulary | | Process | | | | | |
| | | 3.4.10 Establish common cultural values | | Process | | | | | |
| | | 3.4.11 Standardization | | Process | | | | | |
| | | 3.4.12 Adhoc over formal communication | | Process | | | | | |
| | | 3.4.13 Regular cross-team discussions | | Process | | | | | |
| | | 3.4.14 Thematic boards for decision making | | Organization | Could also be viewed as process, but usually have some organizational aligment | | | | |
| | | 3.4.15 Service templates | | Process | | | | | |
| | | 3.4.16 Collaborate on libraries | | Process | | | | | |
| | | 3.4.17 Communicate API changes | | Process | | | | | |
| | | Understanding the system's behavior: | | | | | | | |
| | | 3.4.18 Standardize logging / monitoring / tracing | | Process | | | | | |
| | | 3.4.19 Aggregate logging/monitoring information in a central place | | Operation | | | | | |
| | | 3.4.20 Monitor metrics at different levels | | Operation | | | | | |
| | | 3.4.21 Use dashboards and visualizations | | Operation | | | | | |
| | | 3.4.22 Use a tracing mechanism | | Operation | | | | | |
| | | 3.4.23 Automate anomaly detection and alerting | | Operation | | | | | |
| (C4.1) Scaling Microservice Instances | | | | | | | | | |
| | | 4.1.1 Stateless design | | Architecture | | | | | |
| | | 4.1.2 Auto-scale instances based on metrics | | Operation | | | | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 4.1.3 Load balancing between instances | | Architecture | | | | | | |
| | | 4.1.4 Load balancing by message broker | | Architecture | | | | | | |
| | | 4.1.5 Database clustering and sharding | | Operation | | | | | | |
| (C4.2) Service Autonomy | | | | | | | | | | |
| | | 4.2.1 Self-contained design | | Architecture | | | | | | |
| | | 4.2.2 Storage area isolation per microservice | | Architecture | | | | | | |
| (C4.3) Team Autonomy | | | | | | | | | | |
| | | 4.3.1 Cross-functional teams | | Organization | | | | | | |
| | | 4.3.2 Experiments | | Process | | | | | | |
| | | 4.3.3 Education programs | | Process | | | | | | |
| | | 4.3.4 Support by a task force team | | Organization | | | | | | |
| | | 4.3.5 Use of established patterns | | Process | | | | | | |
| | | 4.3.6 Proximity to domain-knowledge holders | | Organization | | | | | | |
| | | 4.3.7 Local proximity of team members | | Organization | | | | | | |