learning

- Projects Portfolio
- Career Path
- API World
  - API
  - HTTP Responses and Troubleshooting
  - Oauth
  - HTTPS/TLS
  - webhooks
  - websockets
  - CRM
  - Api Troubleshooting

Projects Portfolio

VoIP to Cloud Infrastructure Engineer - Career Roadmap

Your Starting Strengths

- Deep SIP/RTP protocol knowledge
- Experience with Avaya/Cisco voice platforms
- Understanding of real-time system requirements
- Troubleshooting complex distributed voice systems
- Quality of Service (QoS) and network optimization

Career Path Options

1. Cloud-Native Telecom Infrastructure Engineer

Companies: Twilio, Vonage API, Bandwidth, SignalWire, RingCentral
Salary Range: $140K-$200K
Focus: Building containerized, scalable voice platforms

2. Site Reliability Engineer (SRE) - Communications

Companies: Zoom, 8x8, Dialpad, Microsoft Teams, Slack
Salary Range: $140K-$220K
Focus: Ensuring reliability and performance of large-scale communication systems

3. Platform Engineering

Companies: Any large tech company, fintech, or enterprise
Salary Range: $130K-$190K
Focus: Building internal platforms that abstract infrastructure complexity

4. Real-Time Systems Engineering

Companies: Trading firms, gaming companies, IoT platforms
Salary Range: $150K-$250K
Focus: Ultra-low latency distributed systems

12-Month Learning Path

Phase 1: Containerization (Months 1-2)

Goal: Master Docker and container concepts

Skills to Learn:

- Docker fundamentals (images, containers, volumes, networks)
- Docker Compose for multi-container applications
- Container networking deep dive
- Health checks and restart policies
- Image optimization and security

Project 1: Containerized SIP Lab

Goal: Build a complete voice environment in Docker

Components:

- Kamailio SIP proxy in Docker

- FreeSWITCH or Asterisk PBX in Docker

- PostgreSQL for subscriber database

- Homer SIP capture for monitoring
- 2-3 softphones for testing

Deliverables:
- Docker Compose file orchestrating all services
- Documentation on GitHub
- Blog post about challenges faced
- Video demo of making calls through your lab

Skills Demonstrated:
- Multi-container orchestration
- Persistent data management
- Network configuration
- Service discovery

Project 2: Auto-Scaling SIP Proxy

Goal: Build monitoring that triggers container scaling

Components:
- Kamailio with Prometheus exporter
- Grafana dashboards showing CPS, active calls
- Script that scales containers based on load
- Load generator to simulate traffic

Deliverables:
- GitHub repo with all code
- Grafana dashboard JSON
- Performance test results showing scaling

Skills Demonstrated:
- Metrics collection
- Automation scripting
- Performance testing
- Threshold-based scaling

Phase 2: Kubernetes (Months 3-4)

Goal: Deploy and manage stateful telecom apps on Kubernetes

Skills to Learn:

- Kubernetes architecture (pods, services, deployments)
- StatefulSets for stateful applications
- ConfigMaps and Secrets management
- Persistent volumes and storage classes
- Service types (ClusterIP, NodePort, LoadBalancer)
- Ingress controllers
- Network policies
- Pod scheduling and affinity rules

Certification Target: CKA (Certified Kubernetes Administrator)

Project 3: Kubernetes Voice Platform

Goal: Deploy production-grade voice infrastructure on K8s

Components:
- Kamailio deployed as StatefulSet (maintain SIP sessions)
- FreeSWITCH/Asterisk for media
- PostgreSQL with StatefulSet
- Redis for session state
- MetalLB or similar for LoadBalancer
- Cert-manager for TLS
- Monitoring stack (Prometheus, Grafana)

Challenges to solve:
- Session persistence across pod restarts
- SIP UDP load balancing
- Media server IP addressing
- Database failover
- TLS certificate management

Deliverables:
- Helm charts for entire stack
- CI/CD pipeline (GitHub Actions)
- Disaster recovery documentation
- Load testing results (1000+ CPS)

Skills Demonstrated:
- Complex stateful app orchestration
- Production-ready configurations

- High availability patterns
- Performance at scale

Project 4: Multi-Cluster Voice Failover

Goal: Active-passive voice infrastructure across two K8s clusters

Components:
- Two Kubernetes clusters (can use k3s locally)
- Global load balancer (GeoDNS simulation)
- Shared PostgreSQL with replication
- Automated failover scripts
- Health monitoring

Deliverables:
- Complete infrastructure-as-code
- Failover testing documentation
- Recovery time objectives (RTO) measurements
- Runbook for disaster scenarios

Skills Demonstrated:
- Multi-cluster management
- Database replication
- Disaster recovery
- Health check design

Phase 3: Observability (Months 5-6)

Goal: Master monitoring, logging, and tracing for distributed systems

Skills to Learn:

- Prometheus metrics design
- PromQL query language
- Grafana dashboard creation
- Alertmanager configuration
- Log aggregation (ELK or Loki)
- Distributed tracing (Jaeger)
- SLI/SLO/SLA definitions
- Error budgets

Project 5: Complete Observability Stack

Goal: Full observability for voice platform

Components:
- Prometheus for metrics collection
- Custom Kamailio exporter (if needed)
- Grafana dashboards:
  - Call success rate (ASR)
  - Calls per second (CPS)
  - Post-dial delay (PDD)
  - Network quality (jitter, packet loss)
  - Resource utilization
- Loki for log aggregation
- Jaeger for tracing SIP calls end-to-end
- AlertManager with PagerDuty integration

Deliverables:
- Complete monitoring stack as code
- 10+ production-ready dashboards
- Alert rules with severity levels
- On-call runbook
- Capacity planning documentation

Skills Demonstrated:
- Metrics design for business outcomes
- Alert fatigue prevention
- Debugging distributed systems
- SRE practices

Project 6: Chaos Engineering for Voice

Goal: Prove infrastructure resilience through controlled failures

Experiments:
- Kill random SIP proxy pods
- Introduce network latency/packet loss
- Fill database connections
- Exhaust CPU/memory
- Simulate complete AZ failure

Deliverables:
- Chaos test suite (LitmusChaos or similar)
- Results showing graceful degradation
- Incident reports and fixes
- Improved architecture documentation

Skills Demonstrated:
- Resilience engineering
- Failure mode analysis
- System hardening
- Blameless postmortems

Phase 4: Infrastructure-as-Code (Months 7-8)

Goal: Automate everything with code

Skills to Learn:

- Terraform (AWS, Azure, GCP providers)
- Terraform modules and workspaces
- State management and backends
- Ansible for configuration management
- GitOps principles (ArgoCD, Flux)
- Secret management (Vault, SOPS)
- CI/CD for infrastructure

Project 7: Multi-Cloud Voice Infrastructure

Goal: Deploy identical voice infrastructure across AWS and Azure

Components:
- Terraform modules for:
  - Network setup (VPC, subnets, security groups)
  - Kubernetes clusters (EKS, AKS)
  - Databases (RDS, Azure Database)
  - Load balancers
  - DNS configuration

- Ansible playbooks for OS-level config
- GitOps workflow for K8s manifests
- Vault for secrets

Deliverables:
- Complete Terraform codebase
- Module documentation
- Cost analysis per cloud provider
- Deployment automation (single command)
- Drift detection and remediation

Skills Demonstrated:
- Cloud platform expertise
- Cost optimization
- Security best practices
- Automated compliance

Project 8: Self-Service Developer Platform

Goal: Internal platform for deploying SIP endpoints

Components:
- REST API (Go or Python)
- Terraform Cloud or Atlantis
- Web UI for developers
- Automated provisioning:
  - SIP accounts
  - DID assignments
  - Routing rules
  - Firewall rules
- RBAC and approval workflows

Deliverables:
- API documentation (OpenAPI/Swagger)
- Example client implementations
- Admin and user guides
- Security audit results

Skills Demonstrated:
- Platform thinking

- API design
- Developer experience focus
- Security and compliance

Phase 5: Programming & Automation (Months 9-10)

Goal: Write production-quality code for infrastructure automation

Skills to Learn:

- Go or Python (choose one, Go preferred for infrastructure)
- REST API design
- Unit and integration testing
- Code review practices
- Git workflows (branching, PRs)
- Database programming
- Kubernetes Operators/Controllers

Project 9: Custom Kubernetes Operator

```
Goal: Create a SIPProxy Custom Resource Definition (CRD)

Functionality:
- Define SIPProxy resources in YAML
- Operator automatically provisions:
  - Kamailio pods
  - Database schemas
  - ConfigMaps with routing rules
  - Monitoring dashboards
  - DNS entries
- Handle updates and deletions
- Self-healing capabilities

Example Usage:
apiVersion: telecom.example.com/v1
kind: SIPProxy
metadata:
  name: production-proxy
```

```
spec:
  replicas: 3
  region: us-west-2
  database: postgres-prod
  routes:
    - pattern: "^1[0-9]{10}$"
      destination: "pstn-gateway"
```

Deliverables:
- Open source operator on GitHub
- Comprehensive documentation
- Helm chart for operator
- Demo video

Skills Demonstrated:
- Kubernetes internals
- Advanced Go programming
- Operator pattern
- Open source contribution

Project 10: Call Analytics Pipeline

Goal: Real-time CDR processing and analytics

Components:
- Kafka for CDR streaming
- Stream processing (Kafka Streams or Flink)
- TimescaleDB for time-series storage
- Real-time dashboards
- ML-based anomaly detection (optional)
- API for querying call data

Features:
- Fraud detection
- Usage trending
- Quality monitoring
- Cost analysis
- Customer behavior patterns

Deliverables:

- Complete data pipeline
- API documentation
- Sample analytics queries
- Performance benchmarks (millions of CDRs)

Skills Demonstrated:
- Event streaming architecture
- Time-series databases
- API development
- Big data processing

Phase 6: Advanced Topics (Months 11-12)

Goal: Specialized skills that set you apart

Skills to Learn:

- Service mesh (Istio, Linkerd)
- eBPF for observability and security
- Multi-region active-active patterns
- Cost optimization strategies
- Security hardening
- Compliance frameworks (SOC2, HIPAA)

Project 11: WebRTC-SIP Gateway Platform

Goal: Production-ready WebRTC gateway with auto-scaling

Components:
- WebRTC signaling server (Node.js/Go)
- Janus or Jitsi for media
- Kamailio for SIP interworking
- TURN servers for NAT traversal
- Auto-scaling based on concurrent calls
- Geographic routing
- Built-in STUN/TURN infrastructure

Features:
- Browser-to-PSTN calls
- JWT authentication

- Recording capabilities
- Quality monitoring
- Load balancing

Deliverables:
- Complete platform on GitHub
- Demo application
- Performance benchmarks
- Architecture documentation
- Blog series on design decisions

Skills Demonstrated:
- WebRTC expertise
- Full-stack platform design
- Real-time media handling
- Modern telephony architecture

Project 12: Service Mesh for Voice

Goal: Implement Istio for voice microservices

Components:
- Voice platform broken into microservices:
  - SIP routing
  - Authentication
  - Billing
  - Analytics
  - Media control
- Istio service mesh
- mTLS between services
- Advanced traffic management:
  - Canary deployments
  - A/B testing
  - Circuit breaking
  - Retry policies
- Observability through mesh

Deliverables:
- Migration guide from monolith
- Performance comparison

- Security improvements documentation
- Traffic management recipes

Skills Demonstrated:
- Microservices architecture
- Service mesh expertise
- Zero-trust security
- Advanced traffic patterns

Portfolio & Personal Brand

GitHub Profile

Must-Have Repositories:

1. Voice platform on Kubernetes (pinned)
2. Terraform modules for telecom infrastructure (pinned)
3. Custom K8s operator for SIP (pinned)
4. Monitoring/observability examples
5. Code samples in Go/Python
6. Contributions to open-source telecom projects

Profile README:

- Brief intro highlighting telecom → cloud transition
- Links to blog posts and projects
- Certifications and skills
- Contact information

Technical Blog

Article Ideas:

1. "Migrating Legacy PBX to Kubernetes: Lessons Learned"
2. "Auto-Scaling SIP Proxies: When CPS Metrics Matter"
3. "Stateful SIP on Kubernetes: The Hard Parts"
4. "Observability for Voice: Beyond Uptime Monitoring"
5. "Building a Multi-Region Voice Platform with Terraform"
6. "WebRTC-SIP Gateway: Architecture and Performance"

7. "Why Telecom Engineers Make Great SREs"
8. "Database Strategies for High-Volume CDR Storage"

Platforms: Medium, Dev.to, or personal blog

LinkedIn Optimization

Headline:
"Platform Engineer | Building Scalable Cloud-Native Voice Infrastructure | Ex-Avaya/Cisco"

Summary:

- Lead with modern skills (Kubernetes, IaC, Go)
- Mention deep telecom expertise as differentiator
- Link to GitHub and blog
- Highlight specific achievements with metrics

Experience Reframing:

- Before: "Maintained Avaya PBX for enterprise"
- After: "Managed high-availability voice infrastructure serving 10K+ users with 99.99% uptime, handling 500K+ daily call sessions"

Certifications Worth Getting

High Priority

- **CKA (Certified Kubernetes Administrator)** - Most valuable cert
- **AWS Solutions Architect Associate** or **Azure Administrator** - Pick your cloud
- **Terraform Associate** - Shows IaC competency

Medium Priority

- **CKS (Certified Kubernetes Security Specialist)** - After CKA
- **Prometheus Certified Associate** - Good for SRE roles
- **AWS/Azure Advanced Networking** - Leverages existing knowledge

Lower Priority (Nice to Have)

- **CKAD (Certified Kubernetes Application Developer)**
- **HashiCorp Vault Associate**
- **Linux Foundation Certified SysAdmin**

Note: Certifications help with resume screening, but projects prove you can actually do the work.

Job Search Strategy

Resume Tips

Skills Section Format:

```
Infrastructure & Cloud:
- Kubernetes (CKA), Docker, Helm, Service Mesh
- AWS/Azure, Terraform, Ansible, GitOps
- Prometheus, Grafana, ELK/Loki, Jaeger

Programming & Automation:
- Go, Python, Bash scripting
- REST APIs, gRPC, Database programming
- CI/CD pipelines (GitHub Actions, Jenkins)

Telecommunications:
- SIP/RTP protocols, Kamailio, FreeSWITCH, Asterisk
- WebRTC, VoIP QoS, Codec optimization
- High-availability voice architectures
```

Achievement Examples:

- "Architected and deployed Kubernetes-based voice platform handling 1M+ daily calls with 99.95% SLA"
- "Reduced infrastructure costs by 40% through right-sizing and auto-scaling implementation"
- "Built internal platform reducing SIP endpoint deployment time from days to minutes"
- "Implemented observability stack that reduced MTTR by 60%"

Target Companies by Stage

Stage 1: Get Your Foot In (0-6 months)

- Cloud-focused MSPs needing telecom expertise
- UCaaS providers (they understand your background)
- Telecom vendors modernizing (lighter competition)

Stage 2: Level Up (6-12 months)

- Mid-size tech companies with voice products
- SaaS companies with real-time features
- Fintech or gaming companies (value low-latency experience)

Stage 3: Dream Jobs (12+ months)

- FAANG communication teams (AWS Chime, Azure Comms, Google Meet)
- Top-tier UCaaS/CPaaS (Twilio, Vonage, Bandwidth)
- High-frequency trading or gaming infrastructure
- Staff/Principal engineer roles

Interview Preparation

System Design Questions You'll Ace:

- Design a scalable SIP proxy infrastructure
- Build a global call routing system
- Create a real-time analytics pipeline
- Design disaster recovery for stateful services
- Multi-region active-active voice platform

Questions to Prepare:

- Container orchestration strategies for stateful apps
- Tradeoffs between various load balancing approaches
- Handling database failover in distributed systems
- Observability design for microservices
- Cost optimization strategies

Your Unique Value Proposition: "Most cloud engineers have never dealt with sub-100ms latency requirements, stateful UDP protocols, or five-nines uptime for real-time services. My telecom background means I bring reliability and performance thinking that's uncommon in cloud-native teams."

Learning Resources

Books

- **"Kubernetes in Action"** - Marko Luksa (comprehensive K8s)
- **"Site Reliability Engineering"** - Google (SRE principles)
- **"Designing Data-Intensive Applications"** - Martin Kleppmann (systems design)
- **"Terraform: Up & Running"** - Yevgeniy Brikman
- **"Learning Go"** - Jon Bodner

Online Courses

- **KodeKloud** - CKA/CKAD preparation (best for K8s)
- **A Cloud Guru / Linux Academy** - Cloud certifications
- **Udemy: Stephane Maarek's courses** - AWS, Kafka
- **Coursera: SRE Specialization** - Google's SRE program

Communities

- **CNCF Slack** - Kubernetes, Prometheus channels
- **r/kubernetes** - Reddit community
- **VoIP Users Conference** - Telecom community still
- **SRE Weekly Newsletter** - Keep up with industry

Practice Environments

- **Kubernetes the Hard Way** - Kelsey Hightower (deep dive)
- **KillerCoda** - Interactive K8s scenarios
- **Home lab** - 3-node K8s cluster (Raspberry Pis or old PCs)
- **Cloud free tiers** - AWS, Azure, GCP all have free options

---

90-Day Quick Start Plan

Month 1: Foundation

Week 1-2:

- Set up home lab environment
- Complete Docker fundamentals
- Start Project 1 (Containerized SIP Lab)

Week 3-4:

- Finish Project 1
- Begin Kubernetes tutorials
- Write first blog post about Docker journey

Month 2: Kubernetes Deep Dive

Week 5-6:

- CKA study (2 hours/day)
- Deploy simple apps on K8s
- Join CNCF Slack

Week 7-8:

- Take CKA exam
- Start Project 3 (K8s Voice Platform)
- Update LinkedIn with new skills

Month 3: First Applications

Week 9-10:

- Complete Project 3
- Apply to 5 infrastructure/platform roles
- Write blog post about K8s learnings

Week 11-12:

- Start Project 5 (Observability)
- Network on LinkedIn
- Begin informational interviews

Goal by Day 90: Have 2-3 solid projects on GitHub, CKA certification, actively interviewing

---

Success Metrics

3-Month Goals

- [ ] CKA certification obtained
- [ ] 3 projects completed and documented
- [ ] GitHub profile with consistent commits
- [ ] 2 blog posts published
- [ ] Applied to 20+ relevant positions
- [ ] 5 informational interviews completed

6-Month Goals

- [ ] 6 projects completed
- [ ] Contributing to open-source telecom projects
- [ ] Active blog with 5+ posts
- [ ] First infrastructure/platform role offer
- [ ] Cloud certification (AWS/Azure)

12-Month Goals

- [ ] Senior Platform/Infrastructure Engineer role
- [ ] Salary increase of 30-50%
- [ ] Technical influence (conference talk, popular blog)
- [ ] All core projects completed
- [ ] Strong professional network in cloud-native space

Red Flags to Avoid

 Don't:

- Collect certifications without building projects
- Apply to jobs without updating resume/LinkedIn
- Stop at tutorials - always build real things
- Ignore programming - you need scripting at minimum
- Stay in pure legacy PBX roles
- Get discouraged by job rejections (it's a numbers game)

Do:

- Build in public (GitHub, blog posts)
- Network actively (comment, share, connect)
- Apply even if you don't meet 100% of requirements
- Focus on projects that show infrastructure at scale
- Learn from production issues and write about them
- Stay consistent (1-2 hours daily beats weekend binges)

Final Thoughts

Your Competitive Advantage:

You're not starting from zero - you have 10+ years of understanding what production reliability actually means. Most cloud engineers have never:

- Debugged why calls are dropping at 2am
- Dealt with stateful UDP protocols
- Maintained 99.99% uptime for real-time services
- Understood what sub-100ms latency requirements mean

These experiences are VALUABLE. The market needs people who can:

- Build modern infrastructure AND understand telecom
- Design systems that handle real-time traffic at scale
- Bridge the gap between legacy voice and cloud-native

You're Not Transitioning Careers - You're Modernizing Your Skillset

Keep your domain expertise, add modern tooling. That combination is rare and valuable.

Start with Project 1 this week. Small daily progress compounds into career transformation.

Quick Reference

Essential Tools to Install:

- Docker Desktop
- kubectl, helm
- Terraform

- VS Code with extensions
- Git
- AWS/Azure CLI
- Prometheus, Grafana

Daily Habits:

- Code/lab work: 1-2 hours
- Reading: 30 minutes (blogs, docs)
- Networking: 15 minutes (LinkedIn, Slack)

Weekly Habits:

- GitHub commits: 5+ days
- Blog post progress: 1 hour
- Applications: 5-10 jobs
- Learning: New concept or tool

Community Engagement:

- Answer questions in forums
- Share your projects
- Comment on relevant content
- Attend virtual meetups

This roadmap is your guide, not a rigid prescription. Adjust based on your pace, interests, and opportunities. The goal is progression, not perfection.

Next Step: Start Project 1 today. Set up Docker and begin your containerized SIP lab.

Good luck! □□

Career Path

Concrete Learning Path (6-12 months)

Month 1-2: Containerization

- Docker deep dive
- Deploy Kamailio/FreeSWITCH in containers
- Build your own voice lab in Docker Compose

Month 3-4: Kubernetes

- CKA study and certification
- Deploy stateful telecom apps on K8s
- Learn persistent volumes, StatefulSets, load balancing

Month 5-6: Observability

- Instrument a voice platform with Prometheus
- Build Grafana dashboards for SIP metrics
- Set up distributed tracing for call flows

Month 7-8: Infrastructure-as-Code

- Terraform for multi-cloud voice infrastructure
- Ansible/Salt for configuration management
- GitOps workflows

Month 9-10: Programming

- Python or Go for building automation tools
- Build APIs that abstract telecom complexity
- Write operators or controllers for K8s

Month 11-12: Portfolio Projects

- Auto-scaling SIP proxy based on CPS metrics
- Multi-region voice infrastructure with failover

- Internal developer platform for telephony services

Resume Positioning

Reframe your experience:

- "Maintained Avaya PBX" → "Managed high-availability voice infrastructure serving 10K users with 99.99% uptime"
- "Configured SIP trunks" → "Designed and scaled SIP routing infrastructure handling 500K daily call sessions"
- "Troubleshot call quality" → "Performance analysis and optimization of real-time media streams, reduced jitter by 40%"

Red Flags to Avoid

Don't: Just add cloud certs without building things - employers can tell

Don't: Try to jump straight to architect roles without hands-on modern infrastructure experience

Don't: Ignore programming - even infrastructure roles now expect scripting/automation skills

Do: Build public proof of your skills (GitHub repos, blog posts about scaling Kamailio on K8s, etc.)

API World

API

Here’s a clean study table you can add to your guide.

REST Method	Purpose	Description	Common Usage Example
GET	Retrieve data	Used to request or fetch information from a server or API. Does not modify data.	Retrieve customer profile, chat history, ticket status
POST	Create / Send data	Used to submit new data to a server or trigger actions. Often creates new resources.	Start chat session, create ticket, authenticate user
PUT	Update existing data	Used to fully update or replace an existing resource.	Update customer information, modify interaction settings
PATCH	Partial update	Used to update only specific fields of an existing resource.	Change customer status, update phone number only
DELETE	Remove data	Used to delete a resource or remove information from the system.	Delete session, remove user, close/delete record
OPTIONS	Discover supported methods	Used to determine what operations an API endpoint supports. Often used in CORS/preflight checks.	Browser checking allowed API methods
HEAD	Retrieve headers only	Similar to GET but returns headers only, without response body. Useful for validation or testing.	Check if resource exists, validate API availability

---

Example REST API Calls

GET Example

```
GET /api/customers/12345
```

Purpose: Retrieve customer information.

POST Example

```
POST /api/chat/start
```

Purpose: Create/start a new chat session.

PUT Example

```
PUT /api/customer/12345
```

Purpose: Update full customer record.

PATCH Example

```
PATCH /api/customer/12345
```

Purpose: Update only selected fields.

DELETE Example

```
DELETE /api/session/12345
```

Purpose: Delete session or resource.

Easy Interview Explanation

REST APIs

REST APIs allow systems to communicate using standard HTTP methods such as GET, POST, PUT, and DELETE to retrieve, create, update, or remove data.

HTTP Responses and Troubleshooting

HTTP Response Codes & API Troubleshooting Guide

Common HTTP Response Codes

Code	Meaning	What It Usually Means	Common Cause
200	Success	Request completed successfully	API working correctly
201	Created	Resource successfully created	New user, session, or object created
400	Bad Request	API could not process request	Invalid JSON, missing fields, bad formatting
401	Unauthorized	Authentication failed	Invalid token, expired token, missing credentials
403	Forbidden	Access denied	User authenticated but lacks permissions
404	Not Found	Resource or endpoint not found	Wrong URL or API endpoint
405	Method Not Allowed	Wrong HTTP method used	Using GET instead of POST
408	Request Timeout	Request took too long	Slow network or backend delay

Code	Meaning	What It Usually Means	Common Cause
429	Too Many Requests	Rate limit exceeded	Too many API calls
500	Internal Server Error	Backend/server issue	Application crash or server-side failure
502	Bad Gateway	Invalid upstream response	Proxy/load balancer/backend issue
503	Service Unavailable	Service temporarily unavailable	Maintenance or overloaded server

---

Quick API Troubleshooting Flow

Step 1 — Identify the Error Code

Always start with:

- HTTP response code
- error message
- timestamp
- affected endpoint

Example:

```
HTTP/1.1 401 Unauthorized
```

Step 2 — Validate Authentication

Most API issues are:

authentication-related

Check:

- bearer token valid?
- token expired?
- API key correct?
- OAuth issue?
- permissions assigned?

---

401 Unauthorized

Meaning

Authentication failed.

Common Causes

- expired token
- invalid credentials
- missing Authorization header

Example

```
Authorization: Bearer invalid_token
```

Troubleshooting

- regenerate token
- verify OAuth flow
- confirm credentials
- validate headers

---

403 Forbidden

Meaning

Authenticated BUT not authorized.

Common Causes

- missing permissions
- RBAC restrictions
- blocked API access

Troubleshooting

- validate user roles
- confirm API permissions
- verify account access

---

400 Bad Request

Meaning

API request invalid.

Common Causes

- malformed JSON
- missing required fields
- invalid parameters

Example Bad JSON

```
{
"name": "Cesar"
"role": "admin"
}
```

Missing comma causes failure.

Troubleshooting

- validate JSON syntax
- review API documentation
- check required fields
- verify content-type headers

---

404 Not Found

Meaning

Endpoint/resource does not exist.

Common Causes

- incorrect URL
- typo in endpoint
- resource deleted

Troubleshooting

- verify endpoint path
- check API version
- confirm resource exists

---

405 Method Not Allowed

Meaning

Wrong HTTP method used.

Example

Using:

```
GET /api/create-user
```

when API expects:

```
POST /api/create-user
```

Troubleshooting

- verify REST method
- review API documentation

---

429 Too Many Requests

Meaning

API rate limit exceeded.

Common Causes

- excessive API calls
- automation overload

Troubleshooting

- reduce request frequency
- implement retry timers
- review API rate limits

---

500 Internal Server Error

Meaning

Backend application/server failed.

Common Causes

- application crash
- database issue
- backend exception

Troubleshooting

- check backend logs
- identify failed service
- escalate to engineering

---

502 Bad Gateway

Meaning

Gateway/proxy received invalid response.

Common Causes

- load balancer issue
- backend unavailable
- reverse proxy failure

Troubleshooting

- validate backend health
- check proxy/load balancer logs
- verify upstream connectivity

---

503 Service Unavailable

Meaning

Service temporarily unavailable.

Common Causes

- maintenance window
- overloaded system
- service outage

Troubleshooting

- verify service health
- check maintenance alerts
- retry later
- escalate if persistent

Structured Troubleshooting Methodology

1. Reproduce the Issue

Questions:

- Can issue be repeated?
- Is it intermittent?

- Does it affect all users?

---

2. Validate Authentication

Check:

- OAuth flow
- bearer token
- permissions
- API keys

---

3. Validate Request

Check:

- endpoint URL
- HTTP method
- headers
- JSON payload

---

4. Review Response Codes

Use HTTP response code to isolate:

- auth issue
- formatting issue
- backend issue
- permissions issue

---

5. Review Logs

Look for:

- timestamps
- transaction IDs
- correlation IDs
- stack traces

---

6. Validate Connectivity

Check:

- DNS
- firewall
- HTTPS/TLS
- proxies
- load balancers
- ports

---

7. Escalate Properly

Gather:

- screenshots
- logs
- timestamps
- request examples
- reproduction steps

before escalating.

Good Interview Answer

“How do you troubleshoot API issues?”

> “I typically start by identifying the HTTP response code and validating whether the issue is related to authentication, request formatting, permissions, networking, or backend failures. I review request headers, payloads, logs, connectivity, and timestamps to isolate the issue before escalating if necessary.”

Common Interview Tip

NEVER immediately blame the backend.

Good engineers:

- isolate the issue methodically
- validate layers step-by-step
- gather evidence before escalation

That’s what interviewers want to hear.

API World

Oauth

OAuth 2.0 Study Guide

What is OAuth?

OAuth 2.0 is:

an authorization framework

used to securely allow:

- applications
- APIs
- users
- systems

to access resources WITHOUT sharing passwords directly.

Simple Explanation

Instead of giving your password to every application:

```
Application → requests authorization
```

OAuth provides:

temporary access tokens

for secure access.

Real-World Example

Example:

```
You log into an app using Google or Microsoft
```

The app:

- never sees your password
- receives a secure token instead

That process often uses OAuth.

OAuth Main Purpose

OAuth solves:

- secure API authentication
- delegated access
- token-based security
- controlled permissions

Simple OAuth Flow

```
User logs in
      ↓
OAuth server validates identity
      ↓
```

```
Access token issued
      ↓
Application uses token for API calls
```

Important OAuth Components

Component	Purpose
User	Person authenticating
Client Application	App requesting access
Authorization Server	Validates identity and issues tokens
Resource Server	API/backend service
Access Token	Temporary credential used for API access
Refresh Token	Used to obtain new access token

OAuth Tokens

Access Token

Temporary credential used in API requests.

Example:

```
Authorization: Bearer eyJhbGc...
```

Usually:

- short-lived
- expires after some time

Refresh Token

Used to:

request new access token

without forcing user to log in again.

Bearer Token

Bearer token =

token used in Authorization header

Example:

```
Authorization: Bearer abc123xyz
```

Meaning:

> “I already authenticated.”

OAuth vs Bearer Token

OAuth	Bearer Token

Security framework	Actual token
Handles authorization process	Used for API access
Issues tokens	Credential sent in requests

Common OAuth Flow Example

Step 1 — User Authentication

User logs into application.

Step 2 — Authorization Server Validates User

Example:

- Microsoft
- Okta
- Google
- Auth0

Step 3 — Access Token Generated

Example response:

```
{
  "access_token": "abc123xyz",
  "token_type": "Bearer",
  "expires_in": 3600
}
```

Step 4 — API Request Uses Token

```
GET /api/customer
Authorization: Bearer abc123xyz
```

Why OAuth Is Important

OAuth improves:

- security
- scalability
- session control
- API protection
- user management

VERY important in:

- banking

- cloud platforms
- CCaaS
- enterprise APIs

OAuth Benefits

Benefit	Description
No password sharing	Apps never see user password
Secure API access	Token-based authentication
Temporary access	Tokens expire
Permission control	Scoped access
Centralized authentication	SSO/identity provider support

Common OAuth Troubleshooting

Problem 1 — Expired Token

Example response:

```
401 Unauthorized
```

Cause:

- access token expired

Troubleshooting:

- refresh token
- reauthenticate user

Problem 2 — Missing Authorization Header

Missing:

```
Authorization: Bearer token
```

Result:

```
401 Unauthorized
```

Problem 3 — Invalid Token

Possible causes:

- malformed token
- copied incorrectly
- revoked token

Result:

```
401 Unauthorized
```

Problem 4 — Insufficient Permissions

Result:

```
403 Forbidden
```

Meaning:

- authenticated successfully
- lacks required permissions

Problem 5 — Wrong OAuth Scope

OAuth scopes define:

what API access is allowed

Example:

```
read:customers
write:customers
admin
```

If token lacks required scope: API rejects request.

Common OAuth Troubleshooting Flow

Step 1 — Validate Token

Check:

- token present?

- expired?
- malformed?

Step 2 — Validate Authorization Header

Correct format:

```
Authorization: Bearer token
```

Step 3 — Validate Permissions/Scopes

Check:

- API access allowed?
- correct user role?
- proper OAuth scopes?

Step 4 — Validate HTTPS/TLS

OAuth tokens should ONLY travel over:

HTTPS/TLS encrypted connections

Step 5 — Review Logs

Check:

- auth logs
- API logs
- timestamps
- token expiration

OAuth vs API Key

OAuth	API Key
More secure	Less secure
User/session-based	Usually app-based
Temporary tokens	Often static
Permission scopes	Limited control
Enterprise-grade	Simpler authentication

Common Interview Questions

“What is OAuth?”

Good Answer:

> “OAuth 2.0 is an authorization framework that enables secure API access through token-based authentication without exposing user credentials directly.”

“What is a Bearer Token?”

Good Answer:

> “A bearer token is the access token issued during OAuth authentication and used in API requests for authorization.”

“Difference between 401 and 403?”

Code	Meaning
401	Authentication failed
403	Authenticated but not authorized

“How would you troubleshoot OAuth issues?”

Good Answer:

>

> “I would validate the access token, confirm the Authorization header format, verify token expiration and scopes, review authentication logs, and confirm HTTPS connectivity and permissions.”

Important Security Concepts

NEVER expose:

- tokens
- secrets
- credentials

Tokens should always be:

- protected
- encrypted in transit
- short-lived

Easy Memory Trick

OAuth = Security Process

Bearer Token = Access Badge

Example:

OAuth authenticates user

Bearer token grants access

Important Terms To Know

Term	Meaning
OAuth	Authorization framework
Access Token	Temporary API credential
Bearer Token	Token used in requests
Refresh Token	Generates new access token
Authorization Server	Issues tokens
Scope	Permission level
HTTPS/TLS	Secure encrypted communication
401	Authentication failure
403	Permission denied

HTTPS/TLS

HTTPS & TLS Study Guide

What is HTTPS?

HTTPS stands for:

HyperText Transfer Protocol Secure

It is:

encrypted HTTP communication

Used to securely transmit:

- API traffic
- passwords
- tokens
- customer data
- banking information

HTTP vs HTTPS

HTTP	HTTPS
Not encrypted	Encrypted
Insecure	Secure
Data visible in transit	Data protected
Uses port 80	Uses port 443

Why HTTPS Matters

Without HTTPS:

- attackers could intercept traffic
- tokens/passwords could be stolen
- API data could be exposed

VERY important for:

- banking
- cloud platforms
- APIs
- OAuth authentication

What is TLS?

TLS stands for:

Transport Layer Security

TLS is:

the encryption protocol behind HTTPS

Meaning:

```
HTTPS = HTTP + TLS encryption
```

Simple Explanation

HTTP

Data travels in plain text.

HTTPS/TLS

Data is encrypted before transmission.

Example:

```
Customer → Encrypted Traffic → API Server
```

What TLS Protects

TLS provides:

Security Feature	Purpose

Encryption	Protects data confidentiality
Authentication	Verifies server identity
Integrity	Prevents data tampering

Encryption

Encryption converts readable data into:

unreadable encrypted data

Without decryption key: data cannot be understood.

Example

Without TLS:

```
password=MyPassword123
```

could be intercepted.

With TLS:

```
X93kL0sdP2mQ8...
```

encrypted and unreadable.

TLS Handshake (High-Level)

When browser/app connects securely:

Client connects

↓

Server presents certificate

↓

TLS session established

↓

Encrypted communication begins

SSL vs TLS

Older term:

SSL (Secure Sockets Layer)

Modern standard:

TLS

People still casually say:

- SSL certificate
- SSL encryption

But technically: TLS replaced SSL.

Certificates

HTTPS/TLS relies on:

digital certificates

Certificates verify:

- server identity
- trusted domain
- encryption validity

Example

When you open:

```
https://www.glia.com
```

browser checks:

- valid certificate?
- trusted authority?
- secure connection?

Common TLS Components

Component	Purpose
Certificate	Verifies server identity
Public Key	Encrypts data
Private Key	Decrypts data
Certificate Authority (CA)	Trusted issuer
TLS Handshake	Establishes secure session

Why TLS Is Critical For APIs

APIs often transmit:

- bearer tokens
- OAuth tokens

- customer data
- banking information

Without TLS: tokens could be stolen.

OAuth + HTTPS Relationship

OAuth tokens should ONLY travel over:

HTTPS/TLS encrypted connections

Example:

```
https://api.company.com/customers
Authorization: Bearer token
```

Common TLS/HTTPS Troubleshooting

Problem 1 — Certificate Expired

Symptoms:

- browser warning

- API connection failure
- TLS handshake errors

Troubleshooting:

- validate certificate expiration
- renew certificate

Problem 2 — Certificate Not Trusted

Symptoms:

```
Certificate not trusted
```

Causes:

- self-signed certificate
- missing CA chain
- invalid certificate

Problem 3 — TLS Version Mismatch

Example:

- client uses TLS 1.0
- server requires TLS 1.2+

Result: secure connection fails.

Problem 4 — Hostname Mismatch

Certificate issued for:

```
api.company.com
```

but request sent to:

```
test.company.com
```

Result: TLS validation failure.

Problem 5 — Firewall / Proxy Interference

Symptoms:

- HTTPS timeout
- TLS negotiation failure

Check:

- firewall
- proxy
- port 443 access

Common HTTPS/TLS Troubleshooting Flow

Step 1 — Validate URL

Verify:

```
https://
```

not:

```
http://
```

Step 2 — Validate Certificate

Check:

- expiration
- trusted CA
- hostname match

Step 3 — Validate TLS Version

Modern systems typically require:

- TLS 1.2
- TLS 1.3

Step 4 — Validate Port Connectivity

HTTPS typically uses:

port 443

Check:

- firewall
- load balancer
- proxy access

Step 5 — Review Logs

Check:

- TLS handshake errors
- certificate validation failures
- proxy logs

Common Interview Questions

“What is HTTPS?”

Good Answer:

> “HTTPS is secure HTTP communication that uses TLS encryption to protect data transmitted between systems.”

“What is TLS?”

Good Answer:

> “TLS is the encryption protocol that secures HTTPS communications by encrypting traffic, validating server identity, and protecting data integrity.”

“Difference between HTTP and HTTPS?”

HTTP	HTTPS
Unencrypted	Encrypted
Insecure	Secure
Port 80	Port 443

“Why is TLS important for APIs?”

Good Answer:

>

> “TLS protects sensitive API traffic such as OAuth tokens, credentials, and customer data by encrypting communication between systems.”

“What causes TLS failures?”

Common causes:

- expired certificates
- invalid certificates
- TLS version mismatch
- firewall/proxy issues
- hostname mismatch

Important Security Concepts

Never send:

- passwords
- bearer tokens
- OAuth credentials

over:

plain HTTP

Always use:

HTTPS/TLS encrypted communication

Easy Memory Trick

HTTPS = Secure Website/API

TLS = Encryption

Technology Behind HTTPS

Example:

```
HTTPS uses TLS to encrypt traffic
```

Important Terms To Know

Term	Meaning
HTTP	Unencrypted web traffic
HTTPS	Secure encrypted HTTP
TLS	Encryption protocol
SSL	Older predecessor to TLS
Certificate	Verifies server identity
CA	Certificate Authority

Term	Meaning
Port 443	HTTPS secure port
Encryption	Protecting data confidentiality
TLS Handshake	Secure session negotiation

webhooks

Webhooks Study Guide

What is a Webhook?

A webhook is:

an automatic event notification sent from one system to another.

Instead of constantly asking:

> “Did something happen?”

the system:

automatically pushes the event.

Simple Explanation

Example:

```
Customer starts chat
      ↓
Glia sends webhook
      ↓
CRM automatically creates ticket
```

The webhook notifies another system:

in real time.

Webhook = Event-Driven Communication

Webhooks are commonly used for:

- notifications
- integrations
- automation
- workflow triggering

Common Webhook Use Cases

Event	Webhook Action
Customer starts chat	Create CRM ticket
Payment completed	Send confirmation
Call ended	Update reporting system
Agent assigned	Notify supervisor
User authenticated	Trigger workflow

How Webhooks Work

Step 1 — Event Occurs

Example:

```
Chat session started
```

Step 2 — Source System Detects Event

Example:

- Glia
- CRM
- cloud platform

Step 3 — Webhook Sent Automatically

Usually:

HTTP POST request

Example:

```
POST /webhook/chat-started
```

Step 4 — Receiving System Processes Event

Example:

```
CRM updates customer interaction history
```

Example Webhook Payload

```
{
  "event": "chat_started",
  "customerId": "12345",
  "timestamp": "2026-05-20T15:30:00Z"
}
```

Webhook Characteristics

Characteristic	Description
Event-driven	Triggered automatically
Push model	Sends data automatically
Real-time	Immediate notifications
Usually HTTP POST	Most common method
JSON payloads	Common data format

Webhooks vs APIs

VERY IMPORTANT INTERVIEW TOPIC

API	Webhook
Request-based	Event-based
Client asks for data	Server pushes data
Pull model	Push model
Requires polling	Real-time notifications
Client initiates communication	Source system initiates communication

---

Simple Analogy

API

Like calling a restaurant:

> “Is my order ready?”

You continuously ask.

Webhook

Like restaurant texting you:

> “Your order is ready.”

Automatic notification.

API Example (Pull Model)

Client requests:

```
GET /api/chat/status
```

Client repeatedly checks status.

Webhook Example (Push Model)

System automatically sends:

```
POST /webhook/chat-completed
```

No polling required.

Why Webhooks Are Important

Webhooks improve:

- automation
- efficiency
- real-time workflows
- integrations
- scalability

VERY common in:

- CCaaS
- SaaS
- banking
- cloud systems

Common Webhook Use Cases In Glia

Likely webhook events:

- chat started
- chat ended
- agent assigned
- authentication completed
- escalation triggered
- interaction transferred

Common Webhook Troubleshooting

Problem 1 — Webhook Not Received

Possible causes:

- incorrect URL
- firewall issue
- endpoint unavailable
- DNS issue

Problem 2 — Authentication Failure

Webhook endpoint may require:

- API key
- bearer token
- OAuth authentication

Possible result:

```
401 Unauthorized
```

Problem 3 — Invalid JSON Payload

Malformed JSON:

```
{
  "event": "chat_started"
  "customerId": "12345"
}
```

Missing comma causes failure.

Problem 4 — Slow Endpoint Response

If receiving system responds slowly:

- webhook timeout
- retries may occur

Problem 5 — SSL/TLS Issues

Webhook endpoints usually require:

HTTPS/TLS

Problems:

- expired certificate
- invalid certificate
- TLS mismatch

Webhook Troubleshooting Flow

Step 1 — Validate Event Triggered

Did source system generate event?

Check:

- logs
- timestamps
- event history

Step 2 — Validate Webhook URL

Check:

- DNS
- endpoint path
- HTTPS
- port access

Step 3 — Validate Authentication

Check:

- bearer token
- API key
- OAuth credentials

Step 4 — Validate Payload

Check:

- JSON syntax
- required fields
- data types

Step 5 — Review HTTP Response Codes

Code	Meaning
200	Success
401	Authentication failed
403	Permission denied
404	Endpoint not found
500	Receiving server failed

Step 6 — Review Logs

Check:

- webhook delivery logs
- API logs
- server logs
- timestamps

Important Webhook Security Concepts

Because webhooks are inbound requests: they should be protected with:

- HTTPS/TLS
- authentication
- validation
- IP restrictions if needed

Common Interview Questions

“What is a webhook?”

Good Answer:

> “A webhook is an event-driven mechanism where one system automatically sends data to another system when a specific event occurs.”

“Difference between API and webhook?”

Good Answer:

> “APIs typically use a request/response pull model where a client requests data, while webhooks use an event-driven push model where the system automatically sends notifications when events occur.”

“Why use webhooks instead of polling APIs?”

Good Answer:

> “Webhooks provide real-time event notifications and reduce unnecessary polling traffic, making integrations more efficient and responsive.”

“How would you troubleshoot webhook

failures?”

Good Answer:

> “I would validate the event trigger, confirm webhook URL connectivity, verify authentication and HTTPS/TLS configuration, review JSON payload formatting, analyze HTTP response codes, and check delivery logs.”

Easy Memory Trick

API = You Ask

Webhook = System Tells You

Example:

```
API → pull
Webhook → push
```

Important Terms To Know

Term	Meaning
API	Request-based communication
Webhook	Event-driven notification

Term	Meaning
Polling	Repeated API checking
Push Model	Automatic event delivery
Pull Model	Client requests data
JSON Payload	Structured webhook data
HTTPS/TLS	Secure webhook transport
Event-Driven	Trigger-based workflow

websockets

WebSockets Study Guide

What is a WebSocket?

A WebSocket is:

a persistent real-time communication connection between two systems.

Unlike normal REST APIs:

- the connection stays open
- both systems can send data anytime

Simple Explanation

REST API

Client asks for information repeatedly.

Example:

```
"Any new messages?"
"Any new messages?"
"Any new messages?"
```

WebSocket

Connection stays open continuously.

Server pushes updates instantly:

```
"New message received"
```

without client repeatedly asking.

Why WebSockets Exist

WebSockets enable:

real-time communication

Examples:

- live chat
- agent status updates
- notifications
- voice/video signaling
- stock market feeds
- multiplayer gaming

Common WebSocket Use Cases

Use Case	Example
Live Chat	Customer-agent messaging
Agent Presence	Agent online/offline updates
Notifications	Real-time alerts
Voice/Video Apps	Session signaling
Monitoring Dashboards	Live metrics
Collaboration Tools	Real-time updates

How WebSockets Work

Step 1 — Client Connects

Example:

```
Browser → WebSocket Server
```

Step 2 — Connection Upgraded

Initial connection starts using HTTP/HTTPS.

Then upgraded to:

WebSocket protocol

Step 3 — Persistent Connection Established

Connection remains:

continuously open

Both sides can send data anytime.

WebSocket Communication Flow

```
Client ↔ WebSocket Server
```

Bidirectional communication.

REST API vs WebSocket

REST API	WebSocket
Request/response	Persistent connection
Stateless	Stateful

REST API	WebSocket
Client initiates requests	Both sides send data
Polling often required	Real-time updates
Short-lived connection	Long-lived connection

Simple Analogy

REST API

Like sending emails:

- request
- response
- connection closes

WebSocket

Like phone call:

- connection stays active
- both sides talk anytime

Example WebSocket Flow

```
Customer sends message
        ↓
Agent instantly receives message
        ↓
Agent replies immediately
        ↓
```

```
Customer sees reply in real time
```

No repeated API polling needed.

WebSocket URLs

Instead of:

```
https://
```

WebSockets use:

```
ws://
```

Secure WebSockets:

```
wss://
```

Secure WebSockets

WSS = WebSocket Secure

Equivalent of:

HTTPS for WebSockets

Uses:

- TLS encryption
- secure communication

VERY important in:

- banking
- enterprise platforms
- customer interactions

Why WebSockets Matter In Glia

Glia heavily relies on:

- real-time chat
- live agent updates
- voice interactions
- customer engagement

WebSockets are commonly used for:

- live interaction synchronization
- agent presence
- messaging
- session updates

WebSockets vs Webhooks

WebSocket	Webhook
Persistent connection	One-time event notification
Real-time bidirectional	Event-based push
Both sides communicate	Source system pushes
Continuous session	Stateless notification

WebSockets vs REST APIs

REST API	WebSocket
Request/response	Continuous communication
Polling required for updates	Instant updates
Better for CRUD operations	Better for real-time apps

CRUD:

- Create
- Read
- Update
- Delete

Common WebSocket Troubleshooting

Problem 1 — Connection Fails

Possible causes:

- firewall blocking
- proxy issues
- incorrect endpoint
- TLS/certificate issue

Problem 2 — Connection Drops

Possible causes:

- timeout
- network instability
- load balancer session timeout

Problem 3 — TLS/WSS Failure

Symptoms:

- secure connection rejected
- certificate errors

Check:

- TLS certificates
- HTTPS/WSS configuration

Problem 4 — Authentication Failure

Some WebSockets require:

- bearer token
- session authentication
- OAuth validation

Result: connection rejected.

Problem 5 — High Latency

Symptoms:

- delayed chat messages
- slow updates

Possible causes:

- network congestion
- overloaded server
- scaling issues

WebSocket Troubleshooting Flow

Step 1 — Validate Endpoint

Check:

```
ws://
```

or:

```
wss://
```

Step 2 — Validate Connectivity

Check:

- firewall
- proxy

- load balancer
- port access

Step 3 — Validate Authentication

Check:

- bearer token
- OAuth session
- API credentials

Step 4 — Validate TLS/WSS

Check:

- certificate validity
- TLS compatibility
- HTTPS/WSS support

Step 5 — Review Logs

Check:

- WebSocket handshake logs
- disconnect reasons
- timeout events
- session logs

Common Interview Questions

“What is a WebSocket?”

Good Answer:

> “A WebSocket is a persistent bidirectional communication protocol that enables real-time data exchange between systems over a continuously open connection.”

“Difference between REST API and WebSocket?”

Good Answer:

> “REST APIs use a request/response model with short-lived connections, while WebSockets maintain a persistent connection that supports real-time bidirectional communication.”

“Why use WebSockets?”

Good Answer:

>

> “WebSockets are ideal for real-time applications such as live chat, notifications, voice signaling, and agent presence updates because they eliminate the need for continuous polling.”

“Difference between WebSocket and webhook?”

Good Answer:

> “WebSockets maintain a persistent real-time communication session, while webhooks are one-time event notifications triggered when specific events occur.”

“How would you troubleshoot WebSocket issues?”

Good Answer:

> “I would validate endpoint connectivity, verify firewall and proxy access, confirm authentication and TLS configuration, review handshake and disconnect logs, and isolate any network or session timeout issues.”

Important Security Concepts

Secure WebSockets should use:

WSS (WebSocket Secure)

This provides:

- TLS encryption
- secure communication
- token protection

Easy Memory Trick

REST API = Request/Response

Webhook = Event Notification

WebSocket = Live Continuous Conversation

Important Terms To Know

Term	Meaning
WebSocket	Persistent real-time connection
WSS	Secure WebSocket
Bidirectional	Both sides communicate
Persistent Connection	Connection remains open
Polling	Repeated API requests
Real-Time	Instant communication
Handshake	Initial connection setup
Stateful	Maintains session state

CRM

CRM Integrations Study Guide

What is CRM?

CRM stands for:

Customer Relationship Management

A CRM stores customer information and interaction history.

Common CRMs:

- Salesforce
- Zendesk
- ServiceNow
- HubSpot
- Microsoft Dynamics

Why CRM Integrations Matter

CRM integrations allow platforms like Glia to share customer data with business systems.

Example:

```
Glia ↔ CRM ↔ Banking System
```

This helps agents see:

- customer profile
- account details
- previous interactions
- open tickets
- case history

Common CRM Integration Use Cases

Use Case	Description
Customer lookup	Find customer details when interaction starts
Screen pop	Automatically open customer record for agent
Ticket creation	Create case after chat/call
Interaction logging	Save chat/call history in CRM
Status updates	Update ticket or customer journey status
Routing logic	Route customer based on CRM data
Authentication context	Use logged-in user info to identify customer

---

Basic CRM Integration Flow

```
Customer starts chat
      ↓
Glia receives customer ID
      ↓
Glia calls CRM API
      ↓
CRM returns customer data
      ↓
Agent sees customer profile
```

Example CRM API Request

```
GET /api/customers/12345
Authorization: Bearer token
Content-Type: application/json
```

Purpose:

Retrieve customer profile from CRM.

Example CRM API Response

```
{
  "customerId": "12345",
  "name": "Cesar Gonzalez",
  "status": "premium",
  "openCases": 2
}
```

Common Integration Patterns

Pattern	Description
API integration	Systems exchange data using REST APIs
Webhooks	CRM receives event notifications
Embedded widget	Glia embedded inside CRM
Screen pop	Customer record opens automatically
Data sync	Customer data synchronized between systems
SSO	Users authenticate once across systems

CRM Integration With Webhooks

Example:

```
Chat completed
      ↓
Webhook sent to CRM
      ↓
CRM creates case summary
```

Webhook payload:

```
{
  "event": "chat_completed",
  "customerId": "12345",
  "agent": "agent01",
  "summary": "Customer asked about loan status"
}
```

CRM Integration With Screen Pop

Screen pop means:

automatically opening the customer record for the agent

Example:

```
Incoming interaction → CRM opens customer profile
```

This improves:

- agent speed
- personalization
- customer experience

CRM Integration With Routing

CRM data can influence routing.

Example:

```
Customer status = VIP
      ↓
Route to priority queue
```

Other routing examples:

- language
- account type
- product
- region
- open case severity

Common CRM Integration Problems

Problem	Likely Cause
Customer not found	Wrong customer ID or CRM record missing
Screen pop not working	Bad URL mapping or missing customer field
Ticket not created	API failure, bad payload, permissions issue
Data not syncing	Webhook failure or field mapping issue
401 Unauthorized	Invalid or expired token
403 Forbidden	Missing CRM permissions
400 Bad Request	Invalid JSON or missing required fields
Slow response	CRM latency or network issue

CRM Integration Troubleshooting Flow

Step 1 — Understand the Expected Flow

Ask:

> What should happen?
>
> What actually happened?
>
> Where does the flow fail?

Example:

> Expected: Chat ends → CRM ticket created
>
> Actual: Chat ends → No ticket appears

Step 2 — Identify the Integration Type

Check whether the issue involves:

- REST API
- webhook
- SSO
- screen pop
- embedded widget
- data sync

Step 3 — Check Authentication

Validate:

- OAuth token
- bearer token
- API key
- permissions
- scopes

Common errors:

- 401 = authentication problem
- 403 = permission problem

Step 4 — Validate API Request

Check:

- endpoint URL
- HTTP method
- headers
- JSON payload
- required fields

Example:

```
POST /api/cases
Authorization: Bearer token
Content-Type: application/json
```

Step 5 — Validate Field Mapping

Field mapping means matching data between systems.

Example:

Glia Field	CRM Field
customerId	Contact ID
email	Email
interactionId	Case Reference
agentName	Owner
transcript	Case Notes

Common issue:

> Glia sends customerId, but CRM expects contactId

Step 6 — Review Logs

Check:

- API logs
- webhook delivery logs
- CRM logs
- timestamps
- request IDs
- correlation IDs

Step 7 — Validate Network

Check:

- DNS
- firewall
- proxy
- HTTPS/TLS
- allowlists
- rate limits

Step 8 — Test With Known Good Data

Use a test customer record.

Example:

```
Customer ID 12345 exists in CRM
```

Then validate:

- lookup works
- screen pop works
- ticket creation works

Step 9 — Escalate With Evidence

Before escalating, collect:

- timestamps
- request/response examples
- HTTP response codes
- screenshots
- affected users
- correlation IDs

Common Troubleshooting Scenarios

Scenario 1 — CRM Customer Lookup Fails

Possible causes:

- wrong customer ID
- missing record
- incorrect endpoint
- expired token

Troubleshooting:

- verify customer exists in CRM
- test API call manually
- check 401/403/404 errors
- validate field mapping

Scenario 2 — Ticket Creation Fails

Possible causes:

- missing required fields
- invalid JSON
- insufficient permissions
- CRM API outage

Troubleshooting:

- validate JSON payload
- check required fields
- review API response

- confirm CRM service status

Scenario 3 — Screen Pop Not Opening

Possible causes:

- browser pop-up blocked
- bad URL template
- missing customer identifier
- CRM permissions issue

Troubleshooting:

- validate URL format
- confirm customer ID is passed
- test manually in browser
- check agent permissions

Scenario 4 — Webhook Not Updating CRM

Possible causes:

- webhook URL incorrect
- endpoint unavailable
- authentication failure
- TLS certificate issue

Troubleshooting:

- validate webhook delivery logs
- check endpoint availability
- verify HTTPS/TLS
- inspect response codes

Scenario 5 — Slow CRM Response

Possible causes:

- CRM latency
- network delay
- rate limiting
- overloaded integration middleware

Troubleshooting:

- check API response time
- review rate limits
- validate middleware health
- compare with other requests

Good Interview Answer

“How would you troubleshoot a CRM integration issue?”

> “I would first confirm the expected workflow and identify where the failure occurs, whether it is API, webhook, SSO, screen pop, or data sync related. Then I would validate authentication, request payloads, field mapping, response codes, logs, network connectivity, and test with known good records before escalating with clear evidence.”

Key Terms To Know

Term	Meaning

CRM	Customer relationship management system
Screen pop	Auto-open customer record
Field mapping	Matching fields between systems
API integration	Systems exchanging data through APIs
Webhook	Event notification to another system
SSO	Single sign-on
Data sync	Keeping systems updated
Correlation ID	Tracking ID for troubleshooting
Rate limit	Maximum allowed API calls
Middleware	System between two applications

Easy Memory Trick

CRM integration = customer context + automation

It helps agents know:

> Who is the customer?
>
> What happened before?
>
> What should happen next?

Api Troubleshooting

Deep API Troubleshooting Guide

Goal

API troubleshooting is about finding **where the failure is happening**:

```
Client/App → Network → API Gateway → Authentication → Backend Service → Database/External System
```

A good implementation engineer isolates the layer before escalating.

---

1. Start With 4 Basic Questions

Question	Why It Matters
What is expected to happen?	Defines success
What actually happens?	Defines the failure
Is it reproducible?	Separates one-time issue from systemic issue
Who is affected?	Helps identify scope

Example:

Expected: CRM ticket is created after chat ends.

Actual: Chat ends, but no CRM ticket appears.

Scope: One customer environment, all agents.

---

2. Identify the API Direction

Direction	Meaning	Example
Inbound API	Customer system calls platform API	CRM calls Glia API
Outbound API	Platform calls customer system	Glia calls CRM API
Webhook	Event notification sent automatically	Chat ended → CRM ticket
Callback	Response sent after async process	Payment status update

This matters because it tells you **which side to check first**.

---

3. Check the HTTP Method

Method	Troubleshooting Focus
GET	Is the resource ID correct?
POST	Is the payload valid?
PUT	Are all required fields included?
PATCH	Are updated fields allowed?
DELETE	Does the user/token have delete permission?

Example issue:

```
GET /api/create-ticket
```

But the API expects:

```
POST /api/create-ticket
```

Likely result:

405 Method Not Allowed

4. Check the Endpoint

Validate:

- Base URL
- API version
- Path
- Resource ID
- Query parameters
- Environment

Example:

```
https://api.vendor.com/v1/customers/12345
```

Common mistakes:

Problem	Example
Wrong environment	Using sandbox instead of production
Wrong API version	`/v1/` instead of `/v2/`
Typo in endpoint	`/costumers/` instead of `/customers/`
Missing resource ID	`/customers/` instead of `/customers/12345`

5. Check Headers

Important headers:

Header	Purpose
Authorization	Sends bearer token/API key
Content-Type	Format of request body
Accept	Format expected in response

Header	Purpose
x-api-key	API key authentication
Correlation-ID	Request tracking

Example:

```
Authorization: Bearer eyJhbGc...
Content-Type: application/json
Accept: application/json
```

Common issue:

```
Content-Type missing
```

API may not understand the JSON body.

---

6. Check Authentication

Most API failures happen here.

Error	Meaning	Common Cause
401	Not authenticated	Missing/expired/invalid token
403	Authenticated but blocked	Missing permission/scope/role

401 Checklist

Check:

```
Is Authorization header present?
Is token expired?
Is token copied correctly?
Is token for the right environment?
Was token revoked?
```

403 Checklist

Check:

```
Does token have correct scope?
Does user/app have permission?
Is IP allowlisting required?
Is this endpoint restricted?
```

---

7. Check OAuth Scope

Scopes define what the token can do.

Example:

```
read:customers
write:tickets
admin
```

If token only has:

```
read:customers
```

but you call:

```
POST /api/tickets
```

You may get:

```
403 Forbidden
```

---

8. Check the JSON Payload

For POST/PUT/PATCH, inspect payload carefully.

Example:

```
{
  "customerId": "12345",
  "channel": "chat",
  "authenticated": true
}
```

Validate:

Item	Example Problem
Syntax	Missing comma
Required fields	Missing customerId
Data type	`"true"` instead of `true`
Field name	`customerID` instead of `customerId`
Nesting	Object expected, string sent
Array format	Single value sent instead of list

---

9. Understand Status Codes by Category

Code Range	Meaning
2xx	Success
3xx	Redirect
4xx	Client/request issue
5xx	Server/backend issue

Important Codes

Code	Meaning	First Place To Look
200	Success	Validate response body

Code	Meaning	First Place To Look
201	Created	Confirm resource ID returned
204	No Content	Success, but no body
400	Bad Request	Payload/parameters
401	Unauthorized	Authentication
403	Forbidden	Permissions/scopes
404	Not Found	Endpoint/resource
405	Method Not Allowed	HTTP method
408	Timeout	Network/backend delay
409	Conflict	Duplicate/existing resource
415	Unsupported Media Type	Content-Type
422	Validation error	Field validation
429	Rate limit	Too many requests
500	Server error	Backend logs
502	Bad gateway	Proxy/upstream
503	Unavailable	Outage/maintenance
504	Gateway timeout	Backend too slow

10. Troubleshooting by Error Code

400 Bad Request

Likely:

> Invalid payload, missing fields, wrong query parameter

Check:

JSON syntax
Required fields
Field names
Data types
API documentation

401 Unauthorized

Likely:

Authentication failed

Check:

Token present
Token expired
Correct auth method
Correct environment

403 Forbidden

Likely:

Permission issue

Check:

User role
OAuth scopes
API permissions
IP allowlist

404 Not Found

Likely:

Wrong endpoint or resource does not exist

Check:

URL path
API version
Resource ID
Environment

---

409 Conflict

Likely:

Duplicate or conflicting resource

Example:

Trying to create a user that already exists

---

415 Unsupported Media Type

Likely:

Wrong or missing Content-Type

Check:

Content-Type: application/json

---

422 Validation Error

Likely:

Payload is valid JSON but fails business rules

Example:

Phone number format invalid

Required field has invalid value

---

429 Too Many Requests

Likely:

Rate limit exceeded

Check:

Retry-After header

API rate limits

Looping automation

Retry logic

---

500 Internal Server Error

Likely:

Backend application error

Action:

Collect request ID, timestamp, payload sample, and escalate

---

502 / 503 / 504

Likely:

Gateway, service availability, or timeout issue

Check:

- Load balancer
- API gateway
- Backend health
- Maintenance window
- Network latency

---

11. Network Layer Checks

APIs depend on network reachability.

Check:

- DNS resolution
- Firewall
- Proxy
- Port 443
- TLS certificate
- IP allowlisting
- VPN/private connectivity

Common symptoms:

Symptom	Possible Cause
Timeout	Firewall, proxy, backend delay
TLS error	Certificate or TLS mismatch
DNS failure	Wrong hostname
Connection refused	Service down or port blocked

---

12. TLS / Certificate Checks

For HTTPS APIs, validate:

```
Certificate not expired
Hostname matches certificate
Certificate chain trusted
TLS 1.2 or 1.3 supported
```

Common failures:

```
certificate expired
self-signed certificate
hostname mismatch
TLS handshake failed
```

---

13. Rate Limiting

Rate limiting protects APIs from overload.

Common response:

```
429 Too Many Requests
```

Check headers:

```
Retry-After: 60
```

Troubleshooting:

```
Reduce request frequency
Add backoff/retry logic
Check if automation is looping
Review API quota
```

---

14. Timeouts

Timeouts can occur at multiple layers:

```
Client timeout
API gateway timeout
Load balancer timeout
Backend service timeout
Database timeout
```

Ask:

```
How long before failure?
Does it fail always or intermittently?
Is payload large?
Is backend dependency slow?
```

---

15. Correlation IDs

A correlation ID tracks one request across systems.

Example:

```
X-Correlation-ID: abc-123
```

Use it to trace:

```
API Gateway → Auth Service → Backend Service → Database
```

For escalation, always include:

```
correlation ID
timestamp
endpoint
HTTP method
response code
```

---

16. Logs To Review

Log Type	What It Shows
API Gateway logs	Request entry, routing, response code
Auth logs	OAuth/token failures
Application logs	Backend errors
Webhook logs	Delivery attempts
Load balancer logs	Upstream health/timeouts
Firewall/proxy logs	Blocked connections
CRM logs	Customer/ticket integration failures

---

17. Good Troubleshooting Workflow

1. Confirm expected behavior
2. Reproduce issue
3. Identify endpoint + method
4. Check response code
5. Validate auth
6. Validate headers
7. Validate payload
8. Validate network/TLS
9. Check logs/correlation ID
10. Isolate failed layer
11. Escalate with evidence

---

18. Example Scenario

Issue

Chat ends but CRM ticket is not created.

Expected Flow

Chat completed → Webhook sent → CRM API creates ticket

Troubleshooting

Check:

- Was chat_completed event generated?
- Was webhook sent?
- Did CRM receive webhook?
- What HTTP response did CRM return?
- Was token valid?
- Was JSON payload accepted?
- Was ticket required field missing?

Possible findings:

Finding	Meaning
No event generated	Platform workflow issue
Webhook sent, no response	CRM endpoint/network issue
401 from CRM	Auth/token issue
400 from CRM	Payload/field mapping issue
500 from CRM	CRM backend issue

---

19. Another Scenario

Issue

Customer lookup fails during chat start.

Expected Flow

Customer enters → API calls CRM → CRM returns profile

Check:

Customer ID passed correctly?
CRM endpoint correct?
Token valid?
Customer exists?
Field mapping correct?
CRM API response code?

Likely outcomes:

Response	Likely Issue
401	Token expired
403	Missing CRM permission
404	Customer not found
400	Bad customerId format
500	CRM backend failure

---

20. What To Say In Interview

API Troubleshooting Answer

> “I start by confirming the expected workflow and reproducing the issue. Then I identify the API endpoint, HTTP method, response code, headers, authentication method, and payload. Based on the response code, I isolate whether the issue is

> authentication, permissions, request formatting, endpoint, networking, or backend related. I also check logs, timestamps, and correlation IDs before escalating with clear evidence.”

---

21. Implementation Engineer Mindset

Do not just say:

```
The API failed.
```

Say:

```
The POST request to the CRM ticket endpoint is returning 400 because the payload is missing the required caseType field.
```

That shows strong technical ownership.