Azure Service Bus Pub/Sub Architecture Documentation

Table of Contents

Overview

This is a real-time messaging application built on Azure that demonstrates a pub/sub (publish/subscribe) pattern using Azure Service Bus. Users can publish messages through a React frontend, which are then distributed to all connected clients in real-time through WebSocket connections.

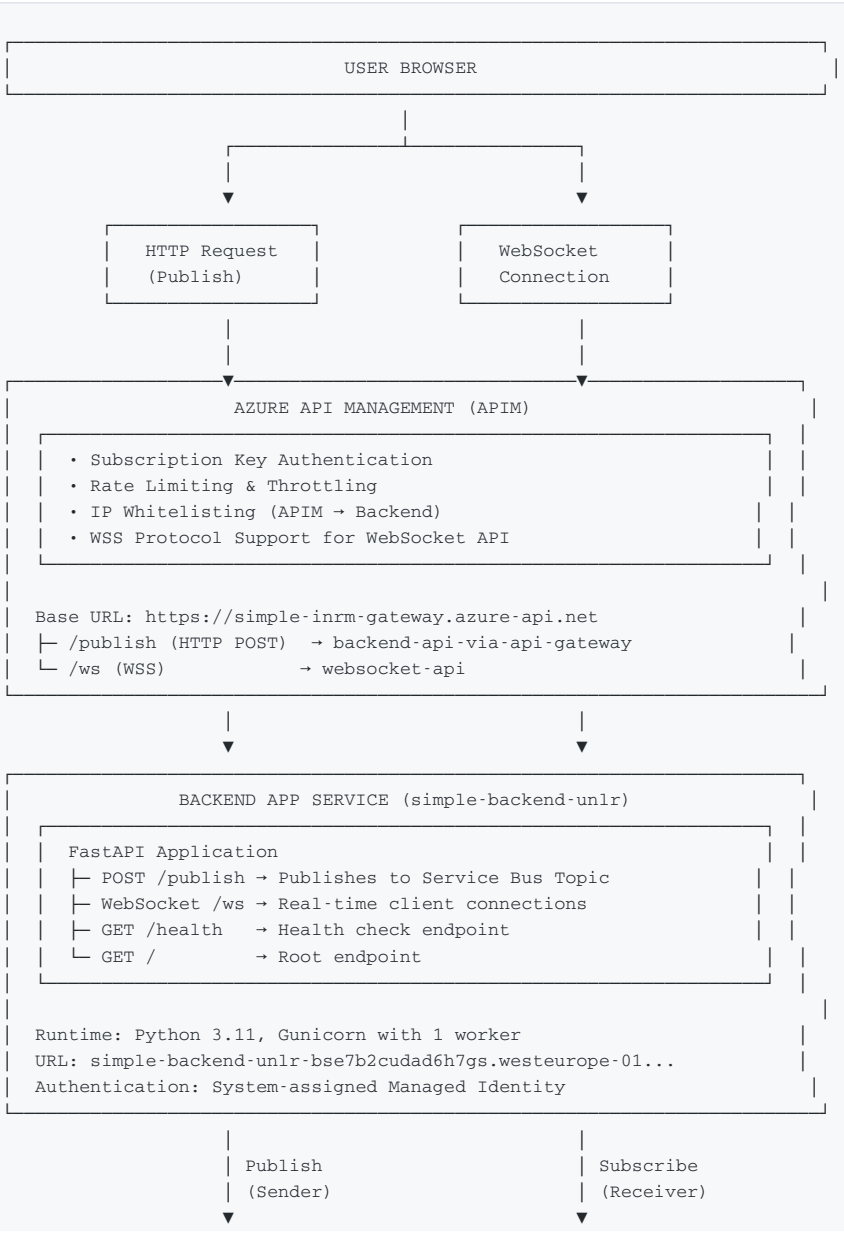
Key Features

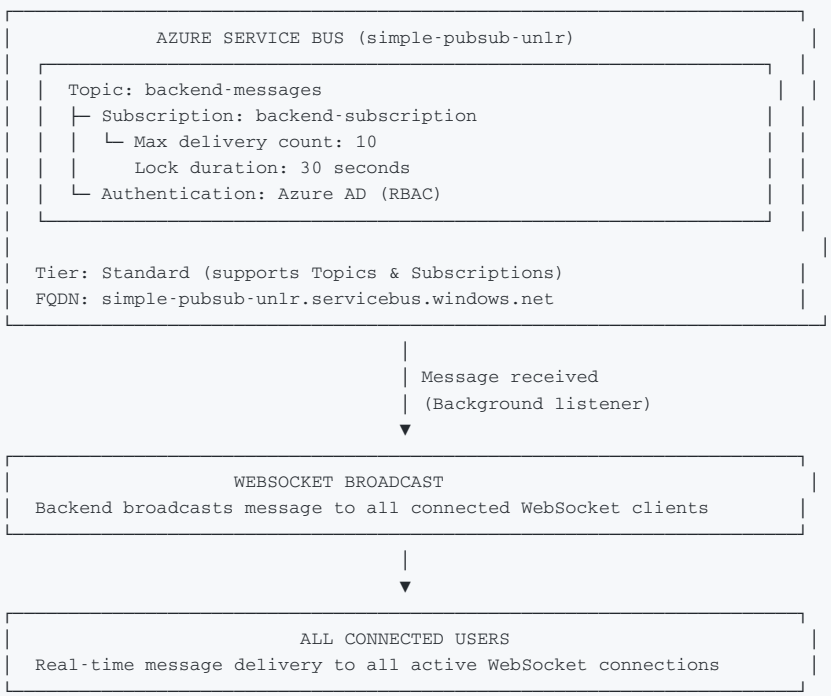
- ✅ **Real-time messaging** via WebSockets
- ✅ **Pub/Sub pattern** using Azure Service Bus
- ✅ **API Gateway** for centralized access control
- ✅ **Azure AD authentication** for Service Bus (Managed Identity)
- ✅ **Automated deployment** via GitHub Actions
- ✅ **Comprehensive logging** for debugging

Technology Stack

Layer	Technology
Frontend	React 18, WebSocket API
Backend	FastAPI (Python), Uvicorn/Gunicorn
Message Broker	Azure Service Bus (Topics & Subscriptions)
API Gateway	Azure API Management
Authentication	Azure AD Managed Identity, APIM Subscription Keys
Hosting	Azure App Service (Linux)
CI/CD	GitHub Actions

Architecture Diagram





Components

1. Frontend Application

File Location: `/frontend`
Hosting: Azure App Service (simple-frontend-unlr)
URL: https://simple-frontend-unlr-g9h4bcgkdtfffxd2.westeurope-01.azurewebsites.net

Description

React-based single-page application that provides the user interface for publishing messages and receiving real-time updates.

Key Features

- Message composition and publishing
- Real-time message display via WebSocket
- Automatic reconnection on connection loss
- APIM subscription key management

Technology Stack

```
{
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "express": "^4.18.2"  // Production server
}
```

File Structure

```
frontend/
├── src/
│   ├── App.js          // Main React component
│   ├── App.css         // Styling
│   └── index.js        // Entry point
├── public/
│   └── index.html      // HTML template
├── server.js           // Express server for production
└── package.json        // Dependencies
```

Configuration

Frontend is configured directly in the code with hardcoded values for APIM endpoints and subscription keys, or they can be set as environment variables in App Service configuration.

How It Works

1. User types a message and clicks "Publish"
2. Frontend sends HTTP POST to `/publish` endpoint through APIM
3. Frontend maintains WebSocket connection to `/ws` endpoint
4. Receives real-time broadcasts when messages arrive from Service Bus
5. Displays messages in the UI

---

2. Backend Application

File Location: `/backend`
Hosting: Azure App Service (simple-backend-unlr)
URL: https://simple-backend-unlr-bse7b2cudad6h7gs.westeurope-01.azurewebsites.net

Description

FastAPI-based Python backend that handles message publishing, Service Bus integration, and WebSocket management.

Key Features

- REST API for message publishing
- WebSocket server for real-time communication
- Service Bus topic publisher
- Service Bus subscription listener (background task)
- Comprehensive logging
- Health check endpoint

Technology Stack

```
Python 3.13
FastAPI 0.104.1
Uvicorn 0.24.0 (built into FastAPI)
Azure Service Bus SDK 7.11.4
Azure Identity SDK 1.15.0
```

File Structure

```
backend/
├── main.py                  // Main application code
├── requirements.txt         // Python dependencies
├── .env.example            // Environment variable template
├── .deployment             // Oryx build configuration
└── runtime.txt             // Python version specification
```

API Endpoints

Method	Path	Description
GET	`/`	Root endpoint, returns status message
GET	`/health`	Health check, returns service status
POST	`/publish`	Publish message to Service Bus
WSS	`/ws`	WebSocket endpoint for real-time communication

Environment Variables

```
# Authentication (choose one method)
AZURE_SERVICEBUS_CONNECTION_STRING=<connection-string>  # OR
AZURE_SERVICEBUS_NAMESPACE_FQDN=simple-pubsub-unlr.servicebus.windows.net

# Service Bus configuration
AZURE_SERVICEBUS_TOPIC_NAME=backend-messages
AZURE_SERVICEBUS_SUBSCRIPTION_NAME=backend-subscription
```

Authentication Methods

Option 1: Connection String (Legacy)

```
client = ServiceBusClient.from_connection_string(CONNECTION_STRING)
```

Option 2: Azure AD Managed Identity (Recommended)

```
credential = DefaultAzureCredential()
client = ServiceBusClient(
    fully_qualified_namespace=NAMESPACE_FQDN,
    credential=credential
)
```

Background Tasks

The backend runs a continuous background task that: 1. Connects to Service Bus subscription 2. Receives messages from the topic 3. Broadcasts received messages to all WebSocket clients 4.

Completes (acknowledges) messages after successful processing

Startup Command

```
python main.py
```

Note: FastAPI with Uvicorn runs in a single process, which is perfect for the in-memory WebSocket connection manager.

---

3. Azure API Management (APIM)

Resource: simple-inrm-gateway
Tier: Developer
Base URL: https://simple-inrm-gateway.azure-api.net

Description

Centralized API gateway that provides security, rate limiting, and protocol translation for the backend services.

APIs Configured

A. HTTP API (backend-api-via-api-gateway) - **Protocol**: HTTPS only - **Path**: `/publish` - **Backend**: https://simple-backend-unlr-bse7b2cudad6h7gs.westeurope-01.azurewebsites.net - **Authentication**: Subscription key required - **Purpose**: Message publishing endpoint

B. WebSocket API (websocket-api) - **Protocol**: WSS (WebSocket Secure) - **Path**: `/ws` - **Backend**: wss://simple-backend-unlr-bse7b2cudad6h7gs.westeurope-01.azurewebsites.net/ws - **API Type**: `websocket` (critical for WSS support) - **Authentication**: Subscription key optional (disabled for simpler connection) - **Purpose**: Real-time communication

Security Features

1. Subscription Keys

```
# Get primary key
az apim subscription show \
  --resource-group uniliver-rg \
  --service-name simple-inrm-gateway \
  --sid master \
  --query "primaryKey" -o tsv
```

2. IP Whitelisting Backend App Service is configured to only accept traffic from APIM's outbound IPs:

```
# APIM IPs that can access backend
52.178.13.125/32
# Or use Service Tag: ApiManagement.WestEurope
```

3. Rate Limiting Configured in APIM policies to prevent abuse.

APIM Configuration Gotchas

Issue: HTTP API cannot use WSS protocol

Solution: Create separate WebSocket API with `api-type: websocket`

Issue: Subscription key in WebSocket query parameters causes FastAPI issues
Solution: Disable subscription requirement for WebSocket API

---

4. Azure Service Bus

Resource: simple-pubsub-unlr
Tier: Standard
Region: West Europe
FQDN: simple-pubsub-unlr.servicebus.windows.net

Description

Message broker that implements the pub/sub pattern using Topics and Subscriptions.

Components

Topic: backend-messages - Receives published messages - Routes messages to all subscriptions - Message retention: 7 days (default) - Max size: 1 GB (Standard tier)

Subscription: backend-subscription - Receives copy of all messages from topic - Max delivery count: 10 - Lock duration: 30 seconds - Dead-letter queue: Enabled (automatic)

Message Flow

```
Publisher (Backend)
    │
    ├─ Creates ServiceBusMessage with JSON payload
    ├─ Sends to Topic "backend-messages"
    │
Topic "backend-messages"
    │
    ├─ Duplicates message to all subscriptions
    │
Subscription "backend-subscription"
    │
    ├─ Stores message until consumer retrieves it
    │
Subscriber (Backend Background Task)
    │
    ├─ Receives message
    ├─ Processes message
    ├─ Broadcasts to WebSocket clients
    └─ Completes (acknowledges) message
```

Authentication & Authorization

Managed Identity Permissions:

```
# Backend App Service has this role assignment
Role: Azure Service Bus Data Owner
Scope: /subscriptions/.../resourceGroups/uniliver-rg/providers/Microsoft.ServiceBus/namespaces/simple-pubsub-unlr
```

Available Roles: - `Azure Service Bus Data Owner` - Full access (send, receive, manage) - `Azure Service Bus Data Sender` - Send only - `Azure Service Bus Data Receiver` - Receive only

Dead Letter Queue

Messages that fail processing (exceeded max delivery count) are automatically moved to the dead-letter queue for manual inspection.

```
# View dead-lettered messages
az servicebus topic subscription show \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --topic-name backend-messages \
  --name backend-subscription \
  --query deadLetterMessageCount
```

---

5. GitHub Actions CI/CD

Location: `.github/workflows/`

Workflows

A. Frontend Deployment (`deploy-frontend.yml`)

```
Triggers:
  - Push to main branch (changes in frontend/)
  - Manual trigger (workflow_dispatch)

Steps:
  1. Checkout code
  2. Setup Node.js 20.x
  3. Install dependencies (npm install)
  4. Build React app (npm run build)
  5. Deploy to Azure App Service
  6. Display deployment summary
```

B. Backend Deployment (`deploy-backend.yml`)

```
Triggers:
  - Push to main branch (changes in backend/)
  - Manual trigger (workflow_dispatch)

Steps:
  1. Checkout code
  2. Setup Python 3.13
  3. Deploy source code to Azure App Service
     - Azure Oryx builds app automatically
     - Creates virtual environment
     - Installs requirements.txt
  4. Display deployment summary
```

Required GitHub Secrets

```
# Publish profiles
AZURE_FRONTEND_PUBLISH_PROFILE  # From: az webapp deployment list-publishing-profiles
AZURE_BACKEND_PUBLISH_PROFILE
```

Azure Oryx Build

Backend uses Azure's Oryx build system: - Detects Python app from `requirements.txt` - Creates virtual

environment at `/home/site/wwwroot/antenv` - Installs all dependencies - Caches packages for faster subsequent builds

Build Settings:

```
SCM_DO_BUILD_DURING_DEPLOYMENT=true
ENABLE_ORYX_BUILD=true
```

Message Flow

Publishing Flow (User → Service Bus)

```
1. User enters message in React frontend
   └─> Message: { content: "Hello", timestamp: "2025-11-30T12:00:00Z" }

2. Frontend sends HTTP POST
   POST https://simple-inrm-gateway.azure-api.net/publish
   Headers:
     Content-Type: application/json
     Ocp-Apim-Subscription-Key: <key>
   Body: { content: "Hello", timestamp: "2025-11-30T12:00:00Z" }

3. APIM validates subscription key
   └─> If invalid: 401 Unauthorized
   └─> If valid: Forward to backend

4. APIM forwards to backend
   POST https://simple-backend-unlr-bse7b2cudad6h7gs.westeurope-01.azurewebsites.net/publish

5. Backend /publish endpoint
   └─> Creates ServiceBusMessage with JSON body
   └─> Uses DefaultAzureCredential (Managed Identity)
   └─> Publishes to Topic "backend-messages"

6. Service Bus Topic
   └─> Receives message
   └─> Duplicates to all subscriptions
   └─> Stores in "backend-subscription"

7. Backend sends immediate confirmation
   └─> Broadcasts to WebSocket clients:
       { type: "publish_confirmation", data: "Message published...", timestamp: "..." }
```

Subscription Flow (Service Bus → Users)

```
1. Backend background task (started on app startup)
   └─> Continuously polls Service Bus subscription
   └─> Max wait time: 5 seconds
   └─> Max messages per receive: 10

2. Service Bus delivers message
   └─> Message locked for 30 seconds
   └─> Delivery count incremented

3. Backend processes message
   └─> Parses JSON body
   └─> Creates WebSocket broadcast message:
       {
         type: "service_bus_message",
         data: "Message from Service Bus: Hello",
         timestamp: "2025-11-30T12:00:00Z",
         original: { content: "Hello", timestamp: "..." }
       }

4. Backend broadcasts to WebSocket clients
   └─> Sends JSON to all connected WebSocket clients
   └─> Logs any failed sends

5. Backend completes message
   └─> Calls receiver.complete_message(msg)
   └─> Removes message from subscription
   └─> If this fails, message is re-delivered (up to max delivery count)

6. Users receive message
   └─> WebSocket onMessage handler fires
   └─> Frontend displays message in UI
```

Authentication & Security

Authentication Layers

```
┌─────────────────────────────────────────┐
│  Layer 1: APIM Subscription Key         │
│  ├─ Frontend → APIM                     │
│  ├─ Header: Ocp-Apim-Subscription-Key   │
│  └─ Validates before reaching backend   │
└─────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────┐
│  Layer 2: IP Whitelisting               │
│  ├─ APIM → Backend                      │
│  ├─ Only APIM IPs allowed               │
│  └─ Blocks direct backend access        │
└─────────────────────────────────────────┘
                    ↓
┌─────────────────────────────────────────┐
│  Layer 3: Azure AD Managed Identity     │
│  ├─ Backend → Service Bus               │
│  ├─ System-assigned Managed Identity    │
│  ├─ Role: Azure Service Bus Data Owner  │
│  └─ No connection strings needed        │
└─────────────────────────────────────────┘
```

Security Best Practices Implemented

✅ **No Secrets in Code** - Connection strings in App Service environment variables only - APIM keys in GitHub Secrets - Managed Identity eliminates Service Bus credentials

✅ **Least Privilege Access** - Backend has only Service Bus Data Owner role - APIM has only deployment permissions - GitHub Actions use publish profiles (limited scope)

✅ **Network Security** - IP whitelisting on backend - HTTPS/WSS only (no plain HTTP/WS) - APIM as single entry point

✅ **Audit Trail** - Comprehensive logging in backend - APIM request logging - Azure Monitor integration

---

Infrastructure Details

Resource Group: uniliver-rg

Region: West Europe

Resources

Resource Type	Name	Purpose
App Service Plan	ASP-uniliverrg-*	Hosting plan for App Services
App Service	simple-frontend-unlr	Frontend hosting
App Service	simple-backend-unlr	Backend hosting
API Management	simple-inrm-gateway	API Gateway
Service Bus Namespace	simple-pubsub-unlr	Message broker

Networking

Frontend App Service: - Outbound IP: Dynamic - Inbound: Public access - Custom domain: Not configured

Backend App Service: - Outbound IP: Dynamic - Inbound: IP restricted to APIM - Allowed IPs: 52.178.13.125/32 or ApiManagement.WestEurope service tag

APIM: - Gateway IP: Public - Outbound IPs: Multiple (for backend calls) - Developer portal: Enabled

Scaling Configuration

Frontend: - SKU: B1 (Basic) - Instances: 1 - Auto-scale: Not configured

Backend: - SKU: B1 (Basic) - Instances: 1 (required for in-memory WebSocket state) - Startup: `python main.py` (single process) - Auto-scale: Not recommended (WebSocket state issues)

APIM: - Tier: Developer - Units: 1

Service Bus: - Tier: Standard - Messaging units: Not applicable (Standard tier)

Deployment

Initial Setup Commands

```
# 1. Create Service Bus resources
az servicebus topic create \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --name backend-messages

az servicebus topic subscription create \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --topic-name backend-messages \
  --name backend-subscription

# 2. Enable Managed Identity
az webapp identity assign \
  --resource-group uniliver-rg \
  --name simple-backend-unlr

# 3. Grant Service Bus permissions
SERVICEBUS_ID=$(az servicebus namespace show \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --query id -o tsv)

PRINCIPAL_ID=$(az webapp identity show \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --query principalId -o tsv)

az role assignment create \
  --assignee $PRINCIPAL_ID \
  --role "Azure Service Bus Data Owner" \
  --scope $SERVICEBUS_ID

# 4. Configure backend
az webapp config appsettings set \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --settings \
    AZURE_SERVICEBUS_NAMESPACE_FQDN="simple-pubsub-unlr.servicebus.windows.net" \
    AZURE_SERVICEBUS_TOPIC_NAME="backend-messages" \
    AZURE_SERVICEBUS_SUBSCRIPTION_NAME="backend-subscription" \
    SCM_DO_BUILD_DURING_DEPLOYMENT=true \
    ENABLE_ORYX_BUILD=true

# 5. Enable WebSockets
az webapp config set \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --web-sockets-enabled true

# 6. Configure IP restrictions
az webapp config access-restriction add \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --rule-name "APIM-Access" \
  --action Allow \
  --ip-address 52.178.13.125/32 \
  --priority 100
```

Automated Deployment

After initial setup, deployments are automatic via GitHub Actions:

```
# Trigger deployment
git add .
git commit -m "Update application"
git push origin main

# Monitor in GitHub
# Go to: https://github.com/your-repo/actions
```

Monitoring & Logging

Backend Application Logs

View real-time logs:

```
az webapp log tail \
  --resource-group uniliver-rg \
  --name simple-backend-unlr
```

Download logs:

```
az webapp log download \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --log-file backend-logs.zip
```

Log Levels

```
INFO  - Normal operations (connections, messages)
DEBUG - Detailed tracing (disabled in production)
ERROR - Failures and exceptions
WARNING - Potential issues
```

Key Log Messages

Startup:

```
Authentication Method: Azure AD (Managed Identity)
Service Bus Namespace FQDN: simple-pubsub-unlr.servicebus.windows.net
✓ Service Bus listener task started
✓ Listening to Service Bus topic 'backend-messages'
```

Message Publishing:

```
Publish request received: {'content': 'Hello', 'timestamp': '...'}
Using Azure AD for publish
✓ Message sent to Service Bus successfully: Hello
✓ Confirmation broadcast to WebSocket clients
```

Message Receiving:

```
Received 1 message(s) from Service Bus
Processing Service Bus message: {"content":"Hello"...
Message processed and completed successfully
```

Errors:

```
ERROR - Service Bus listener fatal error: ...
ERROR - Traceback (most recent call last):
...
```

APIM Logging

Configure in Azure Portal: - API Management → APIs → Select API → Settings → Diagnostic logs - Log all requests/responses - Include headers and body (for debugging)

Service Bus Metrics

Monitor in Azure Portal: - Incoming messages - Outgoing messages - Active connections - Dead-letter message count - Throttled requests

---

Configuration Reference

Frontend Configuration

Frontend endpoints are configured directly in the React code (`App.js`): - APIM Backend URL: `https://simple-inrm-gateway.azure-api.net` - WebSocket URL: `wss://simple-inrm-gateway.azure-api.net/ws` - Subscription keys can be hardcoded or managed separately

Backend Environment Variables

Variable	Value	Purpose
`AZURE_SERVICEBUS_NAMESPACE_FQDN`	`simple-pubsub-unlr.servicebus.windows.net`	Service Bus endpoint (Azure AD)
`AZURE_SERVICEBUS_CONNECTION_STRING`	`Endpoint=sb://...`	Service Bus endpoint (legacy)
`AZURE_SERVICEBUS_TOPIC_NAME`	`backend-messages`	Topic name
`AZURE_SERVICEBUS_SUBSCRIPTION_NAME`	`backend-subscription`	Subscription name
`SCM_DO_BUILD_DURING_DEPLOYMENT`	`true`	Enable Oryx build
`ENABLE_ORYX_BUILD`	`true`	Enable Oryx build

Port Configuration

Service	Port	Protocol
Frontend (dev)	3000	HTTP
Frontend (prod)	8080	HTTP
Backend	8000	HTTP/WSS
APIM Gateway	443	HTTPS/WSS
Service Bus	5671	AMQP over TLS

URL Reference

Component	URL
Frontend	https://simple-frontend-unlr-g9h4bcgkdtfffxd2.westeurope-01.azurewebsites.net
Backend	https://simple-backend-unlr-bse7b2cudad6h7gs.westeurope-01.azurewebsites.net
APIM Gateway	https://simple-inrm-gateway.azure-api.net
APIM HTTP API	https://simple-inrm-gateway.azure-api.net/publish
APIM WebSocket	wss://simple-inrm-gateway.azure-api.net/ws
Service Bus	simple-pubsub-unlr.servicebus.windows.net

Troubleshooting Guide

Common Issues

1. WebSocket connection fails

Symptoms: Frontend shows "Disconnected"

Checks:

```
# Verify WebSocket enabled
az webapp config show \
  --resource-group uniliver-rg \
  --name simple-backend-unlr \
  --query webSocketsEnabled

# Check APIM WebSocket API type
az apim api show \
  --resource-group uniliver-rg \
  --service-name simple-inrm-gateway \
  --api-id websocket-api \
  --query type
```

Solution: Ensure API type is `websocket`, not `http`

---

2. Messages not reaching Service Bus

Symptoms: Publish succeeds but messages not received

Checks:

```
# Check topic exists
az servicebus topic show \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --name backend-messages

# Check subscription exists
az servicebus topic subscription show \
  --resource-group uniliver-rg \
  --namespace-name simple-pubsub-unlr \
  --topic-name backend-messages \
  --name backend-subscription

# Check permissions
az role assignment list \
  --assignee $PRINCIPAL_ID \
  --scope $SERVICEBUS_ID
```

---

3. 403 Forbidden from APIM to Backend

Symptoms: APIM logs show "403 Ip Forbidden"

Checks:

```
# Check IP restrictions
az webapp config access-restriction show \
  --resource-group uniliver-rg \
  --name simple-backend-unlr
```

Solution: Add APIM outbound IP to allowed list

---

4. Backend fails to start

Symptoms: App Service shows unhealthy

Checks:

```
# Check logs
az webapp log tail \
  --resource-group uniliver-rg \
  --name simple-backend-unlr

# Check dependencies installed
# Look for: "WARNING: Could not find virtual environment"
```

Solution: Ensure `SCM_DO_BUILD_DURING_DEPLOYMENT=true`

Additional Resources

- Azure Service Bus Documentation
- Azure API Management Documentation
- Azure Managed Identities
- FastAPI Documentation
- React Documentation

Appendix: Complete Architecture Summary

User Journey: 1. User opens React frontend in browser 2. Frontend establishes WebSocket connection through APIM 3. User types message and clicks "Publish" 4. Message sent via HTTP POST through APIM to backend 5. Backend publishes message to Service Bus topic 6. Service Bus duplicates message to all subscriptions 7. Backend subscriber receives message from subscription 8. Backend broadcasts message to all WebSocket clients 9. All connected users see the message in real-time

Key Design Decisions: - Single process backend with `python main.py` (WebSocket state constraint) - APIM for centralized security and management - Azure AD for Service Bus authentication (no connection strings) - GitHub Actions for automated deployment - Express server for frontend (stable production serving) - Comprehensive logging for debugging - Python 3.13 runtime

Scalability Considerations: - WebSocket connections limited to single backend instance - For multi-instance: Use external state store (Redis, Azure SignalR Service) - Service Bus Standard tier supports higher throughput - APIM Developer tier suitable for development, upgrade for production

Security Posture: - Zero secrets in code repository - Managed Identity for Azure-to-Azure communication - IP whitelisting for backend access - HTTPS/WSS encryption in transit - RBAC for Service Bus authorization