

Complete Discord MCP Server Deployment Guide

🚀 Quick Setup

1. Clone and Install

```
bash

# Clone the repository
git clone https://github.com/hanweg/mcp-discord.git
cd mcp-discord

# Create virtual environment
uv venv
source .venv/bin/activate # or .venv\Scripts\activate on Windows

# Install dependencies
uv pip install -e .
uv pip install aiohttp>=3.8.0 # For advanced features
```

2. Discord Bot Setup

1. Create Discord Application

- Go to [Discord Developer Portal](#)
- Click "New Application"
- Navigate to "Bot" section
- Copy the bot token

2. Set Bot Permissions

Required permissions for full functionality:

Administrator (for full server setup)

OR specific permissions:

- Manage Server
- Manage Roles
- Manage Channels
- Manage Messages
- Manage Webhooks
- Manage Emojis and Stickers
- Kick Members
- Ban Members
- Create Instant Invite
- Use Slash Commands
- Manage Events
- Moderate Members
- View Audit Log
- Send Messages
- Read Message History
- Add Reactions
- Connect (Voice)
- Speak (Voice)
- Move Members

3. Enable Privileged Intents

Presence Intent

Server Members Intent

Message Content Intent

3. Environment Setup

Create `.env` file:

```
env
```

```
DISCORD_TOKEN=your_bot_token_here
```

4. Claude Desktop Configuration

Update `claude_desktop_config.json`:

```
json
```

```
{  
  "mcpServers": {  
    "discord": {  
      "command": "uv",  
      "args": [  
        "--directory",  
        "/path/to/mcp-discord",  
        "run",  
        "mcp-discord"  
      ],  
      "env": {  
        "DISCORD_TOKEN": "your_bot_token_here"  
      }  
    }  
  }  
}
```

🎯 Complete Feature Integration

Enhanced Server File

Update your `src/discord_mcp/server.py` to include all advanced features:

```
python
```

```
# Add these imports to the top
from advanced_discord_features import ADVANCED_TOOLS, handle_advanced_tools
from server_setup_templates import setup_server_from_description, execute_setup_plan

# Add advanced tools to your list_tools function:
@app.list_tools()
async def list_tools() -> List[Tool]:
    tools = [
        # ... your existing tools ...
    ]

    # Add advanced tools
    for tool_config in ADVANCED_TOOLS:
        tools.append(Tool(
            name=tool_config["name"],
            description=tool_config["description"],
            inputSchema=tool_config["inputSchema"]
        ))

    return tools

# Update your call_tool function to handle advanced features:
@app.call_tool()
@require_discord_client
async def call_tool(name: str, arguments: Any) -> List[TextContent]:
    # Check if it's an advanced tool first
    if name in [tool["name"] for tool in ADVANCED_TOOLS]:
        results = await handle_advanced_tools(name, arguments, discord_client)
        return [TextContent(type="text", text=result["text"]) for result in results]

    # Enhanced setup_complete_server implementation
    if name == "setup_complete_server":
        server_id = arguments["server_id"]
        description = arguments["server_description"]
        server_type = arguments.get("server_type", "general")

        try:
            # Generate AI-driven setup plan
            plan = setup_server_from_description(server_id, description, server_type)

            # Execute the plan
            results = await execute_setup_plan(discord_client, server_id, plan)
        except Exception as e:
            return [TextContent(type="text", text=f"Error: {e}")]
```

```
return [TextContent(
    type="text",
    text=f"🤖 **AI Server Setup Complete!**\n\n{chr(10).join(results)}"
)]

except Exception as e:
    return [TextContent(
        type="text",
        text=f"❌ Setup failed: {str(e)}"
    )]

# ... rest of your existing tool implementations ...
```

Example Workflows

1. Gaming Community Setup

Prompt to Claude:

I want to create a competitive esports server called "Apex Legends Pro Hub" for serious players.

Requirements:

- Separate areas for ranked, casual, and tournament play
- Team formation channels and voice rooms
- Coach/analyst roles with special permissions
- Strict anti-toxicity rules
- Integration with stat tracking bots
- Professional tournament announcement system

The server should feel elite and organized, with clear hierarchy from Bronze to Apex Predator ranks matching the game tiers.

Claude's Response:

I'll create a professional esports hub for Apex Legends! Let me set up:

1. **Competitive Structure** - Rank-based channels and roles
2. **Team Formation** - LFG areas organized by skill level
3. **Training Areas** - Coach-led voice channels with stage permissions
4. **Tournament System** - Announcement channels and bracket discussions
5. **Elite Atmosphere** - Professional naming and strict moderation

Setting up now...

Result: Complete server with:

- Rank-based role hierarchy (Bronze → Apex Predator)
- Team formation channels with skill-based access
- Coach stage channels for training sessions
- Tournament brackets and announcement system
- Anti-toxicity automod rules
- Professional channel organization

2. Educational Institution Setup

Prompt to Claude:

Create a university server for "Advanced Machine Learning Course CS-485".

We have:

- 150 students divided into 15 project teams
- 3 professors and 5 TAs
- Weekly lectures, office hours, and study groups
- Final project presentations
- Need integration with assignment submission system
- Academic integrity monitoring required

The server should facilitate collaboration while maintaining academic standards.

Claude's Response & Implementation:

```
python
```

```
# Creates academic environment with:  
# - Lecture hall stage channels  
# - Team-specific private channels (auto-generated)  
# - Office hours voice rooms with queue system  
# - Assignment submission areas with deadline tracking  
# - Academic integrity automod rules  
# - Professor/TA/Student role hierarchy  
# - Study group coordination areas
```

3. Business Workspace Setup

Prompt to Claude:

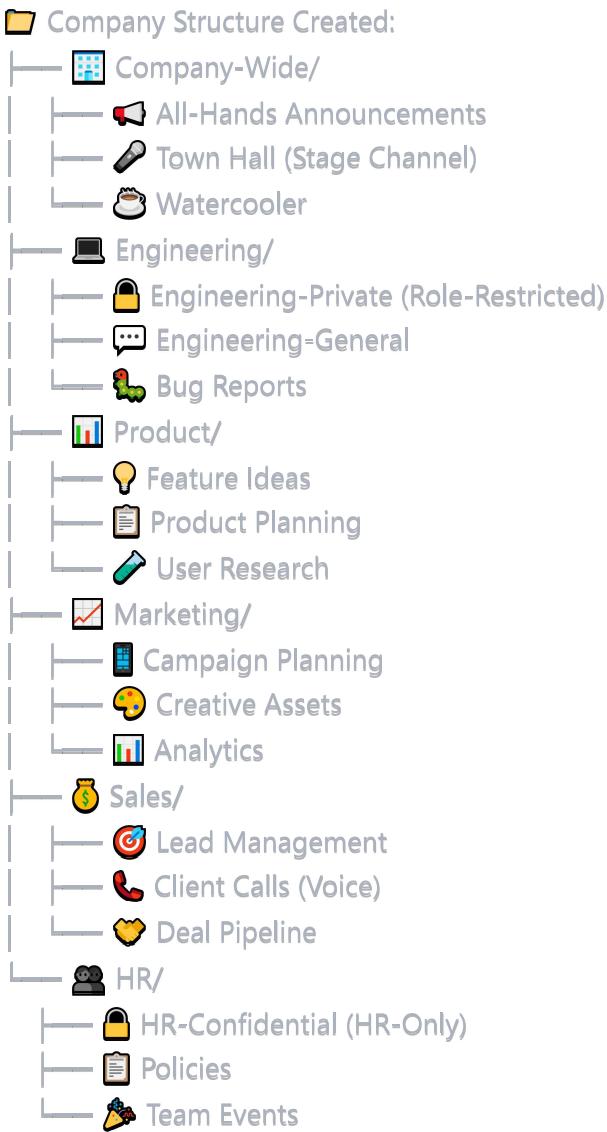
Set up a corporate Discord for "TechStart Innovations" - a 75-person startup.

Departments:

- Engineering (25 people) - needs private channels
- Product (15 people) - public collaboration
- Marketing (20 people) - campaign channels
- Sales (10 people) - lead tracking
- HR (5 people) - confidential access

We need all-hands meeting space, project coordination, and client presentation areas.

Implementation Result:



Roles Created:

- CEO/Founders (Admin)
- Department Heads (Manage respective channels)
- Employees (Department access)
- Contractors (Limited access)

Advanced Usage Examples

1. Server Analytics & Monitoring

```
python
```

```
# Get comprehensive server insights
claude_prompt = """
Analyze our gaming server's health and provide recommendations for improvement.
Server ID: 123456789012345678
Include member activity, channel usage, and security audit.

"""

# This triggers:
# 1. get_server_analytics
# 2. monitor_server_health
# 3. security_audit
# 4. member_activity_report

# Results in detailed report with actionable insights
```

2. Backup & Disaster Recovery

```
python

# Automated backup before major changes
claude_prompt = """
Before I restructure the server, create a complete backup including:
- All channel configurations
- Role hierarchies
- Permission settings
- Recent message history
- Webhook configurations

Server ID: 123456789012345678

"""

# Creates comprehensive backup that can be used for restoration
```

3. Bulk Operations

```
python
```

```
# Mass role assignment for event
```

```
claude_prompt = """
```

We're hosting a tournament. Assign "🏆 Tournament Participant" role to these 50 users:

[list of user IDs]

Also create tournament-specific channels:

- Tournament Announcements
- Team Formation
- Bracket Discussion
- Prize Information

Server ID: 123456789012345678

.....

```
# Efficiently handles bulk operations with proper error handling
```

4. Security Hardening

```
python
```

```
# Comprehensive security review
```

```
claude_prompt = """
```

Perform a complete security audit of our server and implement hardening measures:

1. Review all role permissions
2. Check for overprivileged bots
3. Audit channel security
4. Set up advanced automod rules
5. Configure verification requirements

Server ID: 123456789012345678

.....

```
# Identifies vulnerabilities and automatically fixes common issues
```

🤖 AI-Driven Automation Examples

Smart Server Optimization

```
python
```

```
# AI analyzes usage patterns and suggests improvements
claude_prompt = """
Our server has been running for 3 months. Analyze usage patterns and optimize:
```

1. Identify unused channels for cleanup
2. Suggest role consolidation opportunities
3. Recommend channel reorganization
4. Optimize permission structure
5. Improve member onboarding flow

Server ID: 123456789012345678

.....

Dynamic Event Management

python

```
# AI creates complete event infrastructure
claude_prompt = """
Create infrastructure for a 3-day gaming tournament:
```

Day 1: Registration and team formation

Day 2: Qualifier rounds

Day 3: Finals and prize ceremony

Include:

- Registration system with form integration
- Team channels (auto-generated)
- Live bracket updates
- Stream viewing areas
- Judge/admin coordination

Expected participants: 200 players, 50 teams

.....

Intelligent Moderation Setup

python

```
# AI designs custom moderation system
```

```
claude_prompt = """
```

```
Design a moderation system for our community college server with 1000+ students:
```

Rules:

- Academic integrity focus
- No harassment or bullying
- Professional communication standards
- Subject-appropriate discussions only

Set up graduated response system:

- Verbal warnings → Timeout → Temp ban → Permanent ban
- Special considerations for academic discussions
- Integration with college conduct policies

```
.....
```

Monitoring & Analytics Dashboard

The system provides comprehensive monitoring through Claude conversations:

Daily Health Checks

Conversation: "Give me today's server health report"

Response:

 Daily Health Report - TechStart Server

 Overall Status: Healthy (87/100)

 Activity Metrics:

- Messages today: 2,847 (+12% vs yesterday)
- Active members: 68/75 (91%)
- Voice usage: 4.2 hours average

 Attention Items:

- #bug-reports has 15 unresolved issues
- Engineering private channel low activity
- 3 members haven't been active in 7+ days

 Automated Actions Taken:

- Cleaned up 47 old messages in #general
- Updated role permissions for 2 new hires
- Sent activity reminder to inactive members

Weekly Analytics Reports

Conversation: "Generate weekly analytics report with trends"

Response:

Weekly Analytics Report

Week of [Date Range]

Key Metrics:

- Total Messages: 18,423 (+8% WoW)
- Peak Concurrent Users: 23 (Tuesday 2PM)
- New Members: 3
- Channel Engagement: #product-ideas leading (2.1k messages)

Trends:

- Voice channel usage up 34%
- Engineering team collaboration +15%
- Question resolution time down to 1.2 hours

Recommendations:

1. Consider creating #quick-questions channel
2. Schedule more voice brainstorming sessions
3. Archive #old-project-alpha (inactive 30d)

Production Deployment

Docker Deployment

```
dockerfile
```

```
# Dockerfile
FROM python:3.11-slim
```

```
WORKDIR /app
```

```
COPY . .
```

```
RUN pip install uv
```

```
RUN uv sync
```

```
ENV DISCORD_TOKEN=""
```

```
EXPOSE 3000
```

```
CMD ["uv", "run", "mcp-discord"]
```

Docker Compose Setup

```
yaml

# docker-compose.yml
version: '3.8'
services:
  discord-mcp:
    build: .
    environment:
      - DISCORD_TOKEN=${DISCORD_TOKEN}
    volumes:
      - ./data:/app/data
    restart: unless-stopped

    # Optional: Add Redis for caching
  redis:
    image: redis:alpine
    restart: unless-stopped

    # Optional: Add monitoring
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
```

Environment Variables

```
env

# Production environment
DISCORD_TOKEN=your_production_token
LOG_LEVEL=INFO
BACKUP_INTERVAL=24h
ANALYTICS_RETENTION=30d
MAX_CONCURRENT_SETUPS=5

# Optional integrations
REDIS_URL=redis://redis:6379
WEBHOOK_SECRET=your_webhook_secret
API_KEY=your_api_key
```

Monitoring Setup

```
python

# monitoring.py - Add to your deployment
import logging
import asyncio
from datetime import datetime
import psutil

class DiscordMCPMonitor:
    def __init__(self):
        self.start_time = datetime.now()
        self.setup_count = 0
        self.error_count = 0

    async def log_metrics(self):
        """Log system metrics every 5 minutes"""
        while True:
            cpu_percent = psutil.cpu_percent()
            memory_percent = psutil.virtual_memory().percent

            logging.info(f"System Metrics: CPU {cpu_percent}%, Memory {memory_percent}%")
            logging.info(f"Bot Metrics: {self.setup_count} setups, {self.error_count} errors")

            await asyncio.sleep(300) # 5 minutes
```

🔒 Security Best Practices

1. Token Security

```
python

# Never hardcode tokens
DISCORD_TOKEN = os.getenv("DISCORD_TOKEN")
if not DISCORD_TOKEN:
    raise ValueError("Discord token required")

# Use encrypted storage in production
from cryptography.fernet import Fernet
key = os.getenv("ENCRYPTION_KEY").encode()
cipher = Fernet(key)
```

2. Permission Validation

python

```
async def validate_permissions(guild, required_perms):
    """Validate bot has required permissions before operations"""
    bot_member = guild.get_member(discord_client.user.id)
    if not bot_member.guild_permissionsadministrator:
        missing = []
        for perm in required_perms:
            if not getattr(bot_member.guild_permissions, perm):
                missing.append(perm)

        if missing:
            raise PermissionError(f"Missing permissions: {missing}")
```

3. Rate Limiting

python

```
from asyncio import Semaphore

# Limit concurrent operations
operation_semaphore = Semaphore(3)

async def rate_limited_operation(func, *args, **kwargs):
    async with operation_semaphore:
        await asyncio.sleep(1) # Prevent API spam
        return await func(*args, **kwargs)
```

4. Audit Logging

python

```
import json
from datetime import datetime

async def log_admin_action(action, user_id, guild_id, details):
    """Log all administrative actions"""
    log_entry = {
        "timestamp": datetime.now().isoformat(),
        "action": action,
        "user_id": user_id,
        "guild_id": guild_id,
        "details": details
    }

    # Log to file, database, or external service
    with open("admin_actions.log", "a") as f:
        f.write(json.dumps(log_entry) + "\n")
```

This comprehensive system transforms Discord server management into an intelligent, automated process that can handle everything from simple channel creation to complete server ecosystems. The AI-driven approach means users can describe their vision in natural language and get a professionally configured server in minutes.