🤖 AI Agent Workshop

AI Agenter med Model Context Protocol (MCP)

Lars Søraas
Capgemini
5 November 2025

🎯 Læringsmål

- **Forstå** MCP arkitektur og konsepter
- **Bygge** din egen AI agent med verktøy
- **Utvide** systemet med nye funksjoner
- **Deploye** ved hjelp av Docker containere
- **Lære** beste praksis for produksjon

🗓️ Agenda

1. **Introduksjon til MCP** (15 min)
2. **Arkitektur Oversikt** (15 min)
3. **Hands-on: Utforske Koden** (20 min)
4. **Bygging av Verktøy** (30 min)
5. **Deployment & Testing** (20 min)
6. **Avanserte Funksjoner** (20 min)
7. **Spørsmål & Neste Steg** (10 min)

🧠 Hva er Model Context Protocol?

Model Context Protocol (MCP)

Problemet

- AI modeller er kraftige men **isolerte**
- Trenger tilgang til **sanntidsdata**
- Ønsker å **utføre handlinger** i verden
- Sikkerhet og **standardisering** utfordringer

Løsningen: MCP

- **Standardisert protokoll** for AI-verktøy integrasjon
- **Sikker** og **strukturert** kommunikasjon
- **Utvidbar** arkitektur for alle verktøy

https://modelcontextprotocol.io/specification/2025-06-18 +
https://modelcontextprotocol.io/docs/getting-started/intro

Fordeler med MCP

For utviklere

- 🔧 Standardisert verktøy-grensesnitt
- 🛡️ Innebygde sikkerhetsmønstre
- 📈 Skalerbar arkitektur
- 🔄 Gjenbrukbare komponenter

For AI agenter

- 🌐 Tilgang til eksterne APIer
- 💾 Sanntids data-henting
- ⚡ Utføre handlinger
- 🧠 Forbedret resonnering med kontekst

🏗️ Oversikt arkitektur

📝 Workshop Implementasjon

Forenklet HTTP/REST Transport - lære MCP prinsipper med kjent teknologi

Hva vi bruker:

- ✅ **HTTP endpoints** (`GET /tools`, `POST /weather`)
- ✅ **REST conventions** for enkle API-kall
- ✅ **MCP-compliant format** for tools og responses

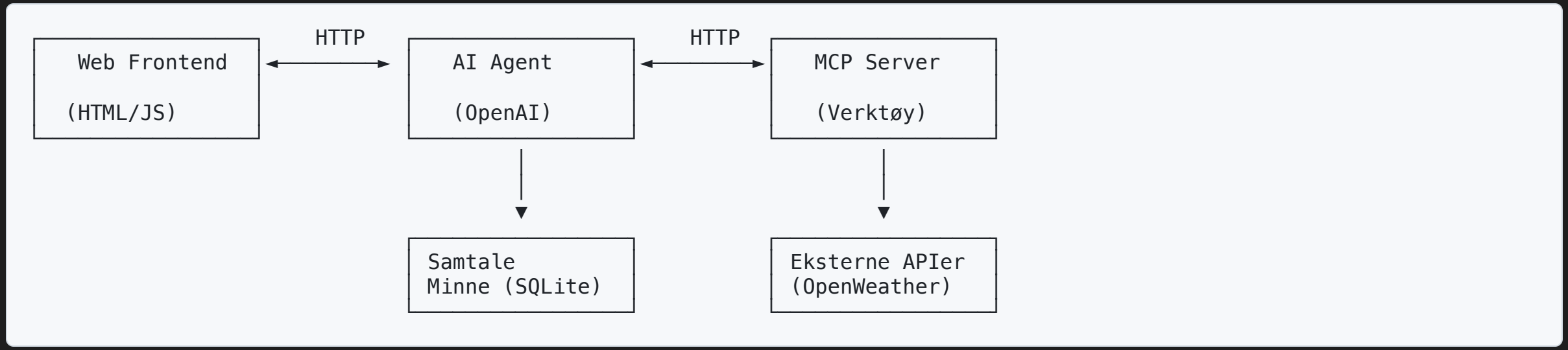
Hva MCP standard bruker:

- 🔄 **JSON-RPC 2.0 protocol** (`tools/list`, `tools/call`)
- 🔄 **Standardiserte metoder** og notifikasjoner
- 🔄 **Multiple transports** (stdio, HTTP SSE, WebSocket)

Hvorfor denne tilnærmingen?

- 📚 **Enklere å lære** - kjente HTTP/REST konsepter
- 🛠️ **Enklere debugging** - curl, Postman, nettleser
- ✅ **Følger MCP dataformat** - tools schema, response structure
- 🎯 **Fokus på konsepter** - ikke protokoll-implementering

Systemarkitektur



Komponenter

💻 MCP Server

- Eksponerer verktøy via HTTP API
- Håndterer ekstern API integrasjon

🤖 AI Agent

- Prosesserer brukerforespørsler med OpenAI
- Kaller MCP verktøy når nødvendig

🌐 Web Grensesnitt

- Enkelt HTML frontend for testing
- Sanntids interaksjon med agent

Dataflyt - Eksempel

Bruker spør: *"Hva er været i Oslo?"*

Oppstart (én gang):
0. **Agent** → **MCP Server**: `GET /tools` (hent tilgjengelige verktøy)

Per forespørsel:

1. **Web** → **Agent**: Brukerforespørsel
2. **Agent** → **OpenAI**: "Bruker vil ha vær for Oslo"
3. **OpenAI** → **Agent**: "Bruk get_weather_forecast verktøy"
4. **Agent** → **MCP Server**: `POST /weather` med parametere
5. **MCP Server** → **OpenWeather API**: Hent værdata
6. **MCP Server** → **Agent**: Vær respons
7. **Agent** → **OpenAI**: "Formater denne værdataen"
8. **Agent** → **Web**: Formatert svar til bruker

🛠️ Etablere utviklingmiljø for workshop

Utviklingsmiljø for workshop

1. Logg inn på din Github konto
2. Lag en *fork* av https://github.com/larssoraas/mcp-ws0511
3. Kryss av Copy the main branch only
4. Velg Code / Codespaces / Create Codespace on...
5. Kopier env.example til .env i Codespace

Du har nå et fiks ferdig utviklingsmiljø!

API nøkkel OpenAI GPT-4.1-mini og OpenWeather

Disse trenger du for å få tilgang til en LMM og værdata

1. Gå til https://github.com/marketplace/models
2. Velg OpenAI GPT-4.1-mini / Use this model / Create Personal Access Token
3. Kopier tokenet - husk - du kan *ikke* få se det på nytt
4. Åpne .env filen i Codespaces og legg tokenet inn i placeholderen for `OPENAI_API_KEY`
5. Register deg gratis på https://openweathermap.org/api og hent ut API Key
6. Legg denne og inn i .env filen

🛠️ Hands-on: Utforske koden

Prosjektstruktur

```
agent/
├── docker-compose.yml          # 🐳 Container orkestrering
├── .env.example                # 🔐 Miljøvariabler
├── services/
│   ├── mcp-server/             # 🔧 Verktøy server
│   │   └── app.py              # ⭐ Hoved server logikk
│   ├── agent/                  # 🤖 AI agent
│   │   ├── app.py              # ⭐ Agent implementasjon
│   │   └── conversation_memory.py
│   └── web/                    # 🌐 Frontend
```

MCP Server - Kodegjennomgang

```python
# Tools endpoint - MCP spec 2025-06-18
@app.get("/tools")
async def list_tools():
    """List available tools according to MCP specification."""
    tools = [{
        "name": "get_weather_forecast",
        "title": "Weather Forecast Provider",
        "description": "Hent værprognose med 5-dagers varsling",
        "inputSchema": {
            "type": "object",
            "properties": {
                "location": {"type": "string", "description": "Navn på by"}
            },
            "required": ["location"]
        },
        "outputSchema": { /* JSON Schema for output validering */ },
        "endpoint": "/weather",
        "method": "POST"
    }]
    return {"tools": tools}

# MCP-compliant response format
@app.post("/weather")
async def get_weather(request: WeatherRequest):
    result = await get_weather_forecast(request.location)
    return {
        "content": [{"type": "text", "text": json.dumps(result)}],
        "structuredContent": result,
        "isError": False
    }
```

Agent - Kodegjennomgang

```python
# services/agent/app.py

# Tools hentes dynamisk fra MCP server ved oppstart
async def load_tools_from_mcp_server(self):
    """Hent tilgjengelige tools fra MCP server."""
    response = await self.http_client.get(f"{self.mcp_server_url}/tools")
    mcp_tools = response.json()

    # Konverter fra MCP format til OpenAI function calling format
    for tool in mcp_tools.get("tools", []):
        openai_tool = {
            "type": "function",
            "function": {
                "name": tool["name"],
                "description": tool["description"],
                "parameters": tool["inputSchema"]
            }
        }
    self.tools.append(openai_tool)
    # Lagre endpoint mapping for senere bruk
    if "endpoint" in tool:
        self.tool_endpoints[tool["name"]] = {
            "endpoint": tool["endpoint"],
            "method": tool.get("method", "POST")
        }
```

🎯 MCP Arkitektur

MCP-Compliant Implementation

- **Følger MCP spec 2025-06-18** for tools og responses
- **Tools med inputSchema og outputSchema** for validering
- **Strukturerte responses** med content[], structuredContent, isError
- **Agent henter tools automatisk** ved oppstart

MCP Response Format

```
{
  "content": [{"type": "text", "text": "..."}],
  "structuredContent": { /* actual data */ },
  "isError": false
}
```

Enklere utvikling

- **Kun endre MCP server** for å legge til nye verktøy
- **Agent håndterer MCP responses** automatisk
- **Type-safe** med JSON Schema validering

https://modelcontextprotocol.io/specification/2025-06-18/server/tools

Flyt - Verktøykall

```
# Når OpenAI vil bruke et verktøy
if response_message.tool_calls:
    for tool_call in response_message.tool_calls:
        tool_name = tool_call.function.name
        tool_args = json.loads(tool_call.function.arguments)

        # Kall MCP server - håndterer MCP-compliant response
        tool_result = await self.call_mcp_tool(tool_name, tool_args)

        # Parse MCP response format
        # result har: content[], structuredContent, isError
        # Ekstraherer structuredContent eller content[0].text

        # Legg til resultat i samtalen
        messages.append({
            "role": "tool",
            "tool_call_id": tool_call.id,
            "content": tool_result  # JSON string fra structuredContent
        })
```

🔧 Hands-on: Bygging av verktøy

Labøvelse 1: Utforsk nåværende verktøy

La oss undersøke værverktøyet og MCP arkitekturen

```
# Start systemet
docker-compose up -d

# Test tools endpoint - se tilgjengelige verktøy
curl -s "http://localhost:8000/tools" | python3 -m json.tool

# Test MCP server direkte
curl -X POST "http://localhost:8000/weather" \
  -H "Content-Type: application/json" \
  -d '{"location": "Oslo"}'

# Test via agent
curl -X POST "http://localhost:8001/query" \
  -H "Content-Type: application/json" \
  -d '{"query": "Hva er været i Bergen?"}'
```

Labøvelse 2: Legg til nytt verktøy

La oss lage et "Tilfeldig Fakta" verktøy i MCP serveren

Steg 1: Legg til faktum-endpoint i MCP Server

```
# I services/mcp-server/app.py

class FactRequest(BaseModel):
    category: str = "general"

@app.post("/fact")
async def get_random_fact(request: FactRequest):
    """Få et tilfeldig interessant faktum."""
    facts = {
        "general": ["Honningbien produserer mat spist av mennesker.",
                    "Bananer er bær, men jordbær er ikke det."],
        "space": ["En dag på Venus er lengre enn året sitt.",
                  "Saturn ville flyte i vann."]
    }

    import random
    fact = random.choice(facts.get(request.category, facts["general"]))
    result = {"category": request.category, "fact": fact}

    # MCP-compliant response format
    return {
        "content": [{"type": "text", "text": json.dumps(result)}],
        "structuredContent": result,
        "isError": False
    }
```

Labøvelse 2: Oppdater tools manifest

Steg 2: Legg til i /tools endpoint

```python
# I services/mcp-server/app.py, oppdater list_tools():

@app.get("/tools")
async def list_tools():
    tools = [
        {
            "name": "get_weather_forecast",
            "description": "Hent værprognose for en destinasjon",
            "inputSchema": { /* ... */ }
        },
        {
            "name": "get_random_fact",
            "title": "Random Fact Provider",
            "description": "Få et tilfeldig interessant faktum",
            "inputSchema": {
                "type": "object",
                "properties": {
                    "category": {
                        "type": "string",
                        "description": "Faktakategori (general, space)",
                        "default": "general"
                    }
                }
            },
            "outputSchema": {
                "type": "object",
                "properties": {
                    "category": {"type": "string"},
                    "fact": {"type": "string"}
                }
            },
            "endpoint": "/fact",
            "method": "POST"
        }
    ]
    return {"tools": tools}
```

Labøvelse 2: Oppdater agent-mapping

Steg 3: Test det nye verktøyet

```
# Bygg på nytt og restart (agent henter tools ved oppstart)
docker-compose build mcp-server travel-agent
docker-compose up -d

# Test at tools er oppdatert
curl -s "http://localhost:8000/tools" | python3 -m json.tool

# Test det nye verktøyet
curl -X POST "http://localhost:8001/query" \
  -H "Content-Type: application/json" \
  -d '{"query": "Fortell meg et interessant faktum om verdensrommet"}'
```

Forventet resultat???

Labøvelse 2.5: Få agenten til å bruke alle verktøy

Test på nytt - nå skal alle verktøy fungere!

Labøvelse 3: Ekte API integrasjon

La oss legge til et ekte Nyhets API verktøy

```python
# I services/mcp-server/app.py
import httpx

class NewsRequest(BaseModel):
    topic: str
    language: str = "no"

@app.post("/news")
async def get_news(request: NewsRequest):
    """Få nylige nyheter om et emne."""
    api_key = os.getenv("NEWS_API_KEY")

    async with httpx.AsyncClient() as client:
        response = await client.get(
            "https://newsapi.org/v2/everything",
            params={
                "q": request.topic,
                "language": request.language,
                "apiKey": api_key
            }
        )
    news_data = response.json()
    articles = [
        {"title": article["title"], "url": article["url"]}
        for article in news_data.get("articles", [])[:3]
    ]

    result = {"topic": request.topic, "articles": articles}
    # MCP-compliant response
    return {
        "content": [{"type": "text", "text": json.dumps(result)}],
        "structuredContent": result,
        "isError": False
    }
```

🐋 Deployment & Test

Docker compose kommandoer

Utviklingsarbeidsflyt

```
# Start fra bunnen av
docker-compose up --build

# Stopp alt
docker-compose down

# Bygg spesifikk tjeneste på nytt
docker-compose build mcp-server

# Se logger
docker-compose logs -f travel-agent

# Sjekk helse
curl http://localhost:8000/health
```

Miljøoppsett

1. Kopier miljøfil

```
cp .env.example .env
```

2. Legg til API nøkler

```
# .env
OPENAI_API_KEY=your-openai-api-key-here
OPENWEATHER_API_KEY=your-openweather-api-key-here
NEWS_API_KEY=your-news-api-key-here  # Hvis du bruker news verktøy
```

3. Bygg og start

```
docker-compose up --build
```

Teststrategi

1. Enhetstesting (Individuelle verktøy)

```
# Test MCP server endepunkter direkte
curl -X POST "http://localhost:8000/tools/get_weather_forecast" \
  -H "Content-Type: application/json" \
  -d '{"location": "Oslo"}'
```

2. Integrasjonstesting (Full flyt)

```
# Test gjennom agent
curl -X POST "http://localhost:8001/query" \
  -H "Content-Type: application/json" \
  -d '{"query": "Hva er været i dag?"}'
```

3. Webtesting

Åpne http://localhost:8080 i nettleser

Tips - Debugging

Vanlige problemer & løsninger

🔴 Container vil ikke starte

```
docker-compose logs service-name
```

🔴 API kall feiler

```
docker-compose exec travel-agent env | grep API
```

🔴 Verktøy ikke gjenkjent

```
curl http://localhost:8001/health
```

🚀 Avanserte Funksjoner

Avanserte funksjoner - vanlige mønstre

1. Tilstand og historikk

- Opprettholde kontekst mellom kall
- Brukerpreferanser og historie
- Sesjonshåndtering

2. Asynkrone operasjoner

- Langvarige oppgaver
- Bakgrunnsprosessering
- Fremdriftssporing

3. Flertrinns arbeidsflyt

- Verktøy-kjeding
- Betinget logikk
- Feilhåndtering og retry

Sikkerhetsbetraktninger

Inputvalidering

```python
from pydantic import BaseModel, validator

class SecureRequest(BaseModel):
    location: str

    @validator('location')
    def validate_location(cls, v):
        if len(v) > 100:
            raise ValueError('Lokasjon for lang')
        # Legg til mer validering...
        return v
```

Håndtering - API nøkler

```python
import os

def get_api_key(service: str) -> str:
    key = os.getenv(f"{service.upper()}_API_KEY")
    if not key:
        raise ValueError(f"Mangler {service} API nøkkel")
    return key
```

Deployment - Produksjon

Skalerbarhet

- **Lastbalansering** med flere agent instanser
- **Database** for delt "minne"
- **Caching** for ofte brukte verktøy-resultater

Overvåkning

- **Helsesjekk** og oppetid overvåkning
- **Logging** ELK, Graylog, Splunk...
- **Metrics** Prometheus,Grafana
- **Feilsporing** (Sentry)

Sikkerhet

- **API rate limiting**
- **Input valitering og vasking**
- **HTTPS terminering**
- **Håndtering av hemmeligheter**

Utvidelse av arkitektur

Legg til nye funksjoner

🧠 Minne

- Vektordatabaser for semantisk søk
- Kunnskapsgrafer for relasjoner
- Langsiktig læring og tilpasning

🔗 Integrasjon

- Webhooks for sanntid oppdateringer
- Meldingskøer for asynkron prosessering
- Hendelsesdrevet arkitektur

🌐 Multi-Modal atøtte

- Bildeanalyse verktøy
- Lydprosessering
- Tolking av video

💡 Workshop Øvelser

Øvelse 1: Værforbedring

Grad: Nybegynner

Forbedre værverktøyet til å inkludere:

- UV-indeks informasjon
- Luftkvalitet data
- Soloppgang/solnedgang tider

Tips: OpenWeatherMap API gir all denne dataen i responsen!

Øvelse 2: Kalkulatorverktøy

Grad: Nybegynner

Lag et kalkulator verktøy som kan:

- Utføre grunnleggende matematiske operasjoner
- Håndtere komplekse uttrykk
- Vise steg-for-steg løsninger

```
# Verktøy idé
@app.post("/tools/calculate")
async def calculate(request: CalculationRequest):
    # Din implementasjon her
    pass
```

Øvelse 3: Minne-Aktivert Chat

Grad: Mellomnivå

Utvid agenten til å huske:

- Brukerpreferanser (favoritt byer, enheter)
- Tidligere samtaler
- Personaliserte anbefalinger

Filer å modifisere:

- `conversation_memory.py`
- Logikk for samtale med agent

Øvelse 4: Orkestrering - Arbeidsflyt

Grad: Avansert

Lag en reiseplanlegging arbeidsflyt:

1. Få vær for destinasjon
2. Finn nærliggende attraksjoner
3. Sjekk kalender tilgjengelighet
4. Send e-post sammendrag

Krever: Flere API integrasjoner + logikk for arbeidsflyt

🎓 Læringspunkter

Hva du har lært

🧠 Konsepter

- Model Context Protocol grunnleggende
- AI agent arkitektur mønstre
- Verktøy-basert AI system design

🛠️ Ferdigheter

- Bygge MCP-kompatible verktøy
- Integrere eksterne APIer sikkert
- Docker containerisering for AI systemer

🚀 Beste praksis

- Strukturert feilhåndtering
- Sikkerhetsbetraktninger
- Testing strategier

Neste Steg

📚 Fortsett å lære

- Utforsk MCP spesifikasjonen i dybden
- Studer avanserte AI agent mønstre
- Lær om vektor databaser og RAG

🔧 Bygg mer

- Lag industri-spesifikke verktøy
- Implementer produksjon overvåkning
- Skaler til multi-tenant arkitektur

🌐 Community

- Bli med i MCP utvikler community
- Bidra til open source verktøy
- Del implementasjoner

Ressurser

📖 Dokumentasjon

- Model Context Protocol Docs
- OpenAI API Reference
- Docker Compose Guide

🛠️ Verktøy & Biblioteker

- FastAPI Dokumentasjon
- Pydantic for Data Validering
- HTTPX for Async HTTP

💻 Kildekode

- Workshop repository: `github.com/your-repo/agent-workshop`
- Eksempel implementasjoner
- Utvidet verktøy bibliotek

🙋‍♂️ Spørsmål & diskusjon

Takk for deltakelsen!

Lars Søraas
📧 lars.soraas@capgemini.com
🐙 github.com/larssoraas

🚀 Lykke til videre!

Husk:

- Start enkelt, iterer raskt
- Sikkerhet først, alltid
- Dokumentasjon er din venn
- Community er her for å hjelpe