Tutorial 33: Slack Bot Integration with ADK

> **Difficulty:** intermediate-advanced
> **Reading Time:** 1.5 hours
> **Tags:** ui, slack, python, bot, messaging
> **Description:** Build intelligent Slack bots with Google ADK for team support, knowledge base search, and enterprise automation.

This tutorial has been verified against official Slack Bolt Python SDK (v1.26.0 - verified October 2025), Google ADK patterns, and production deployment best practices.

Estimated Reading Time: 50-60 minutes
Difficulty Level: Intermediate to Advanced
Prerequisites: Tutorial 1-3 (ADK Basics), Python 3.9+, Slack workspace admin access

---

Table of Contents

Why Slack + ADK? (Real-World Value)

The Problem You're Solving

Teams waste **3-4 hours per day** switching between tools to answer questions:

- "What's our vacation policy?"
- "How do I reset my password?"
- "Which project should I focus on?"

Developers waste context switching time. Support teams field repetitive questions. Knowledge lives in scattered places.

The ADK Solution

With Slack + ADK, you build an **intelligent bot that lives where your team already works**:

```
Without Bot:
User → Google Docs → Notion → Wiki → Email support team → Wait 4 hours

With Slack Bot:
User: @Support Bot help with expense reports
Bot:   (instant response with the exact policy + ticket creation option)
```

Real-World Learning Gains

By the end of this tutorial, you'll be able to:

- ✔️ **Build intelligent Slack bots** that understand context and respond in real-time
- ✔️ **Integrate ADK agents** with Slack Bolt for production-grade bots
- ✔️ **Manage conversation state** across threads and DMs

- ✔️ **Deploy to Cloud Run** safely with secrets and monitoring
- ✔️ **Handle 100+ concurrent users** without manual scaling
- ✔️ **Create tools** that execute real business logic (ticket creation, knowledge base search)

Who Should Use This?

Role	Why Slack + ADK?
Platform Engineers	Build internal developer tools that feel native to workflows
DevOps Teams	Create incident response bots that execute runbooks in Slack
Product Managers	Deploy analytics dashboards and decision-making tools
Support Teams	Automate FAQ responses and ticket triage
HR/People Teams	Build onboarding bots and policy finders

Why Not Web UI?

When to choose **Slack** vs **Web UI** (Tutorial 30):

Feature	Slack Bot	Web UI
Setup	Easy (in team's workflow)	Requires URL sharing
Adoption	Native (9/10 usage)	Low friction (2/10 usage)
Context	Rich (user, channel, thread)	Limited (just user)
Public	Internal team tool	External customer-facing
Mobile	Works on Slack Mobile	Needs responsive design

Use Slack for internal team tools. Use Web UI for customer-facing apps.

What You'll Learn

By completing this tutorial, you'll understand:

Concepts:
- How Slack bots integrate with ADK agents
- Socket Mode (development) vs HTTP Mode (production)
- Session state and conversation threading
- Tool integration and execution flows

Skills:
- Configure Slack apps and OAuth scopes
- Build event handlers for mentions and DMs
- Create callable tools that agents execute
- Deploy to Cloud Run with secrets
- Monitor and troubleshoot production bots

Code:
- Working Slack bot with 100+ lines of production code
- Two callable tools (knowledge base search, ticket creation)
- Complete test suite (50 tests)
- Ready-to-deploy Docker configuration

Overview

What You'll Build

In this tutorial, you'll build a **team support assistant Slack bot**:

```
┌─────────────────────────────────────────────┐
│  Team Support Bot (@support-bot)            │
│  ├─ Intelligent responses                   │
│  ├─ Knowledge base search (tool)            │
│  ├─ Support ticket creation (tool)          │
│  ├─ Thread-aware conversations              │
│  └─ Production deployment ready             │
└─────────────────────────────────────────────┘
```

This bot will:

1. **Listen** for mentions like `@Support Bot how do I reset my password?`
2. **Search** your knowledge base for relevant articles
3. **Create** support tickets when issues need human review
4. **Respond** with formatted messages in Slack threads

Architecture: Three Layers

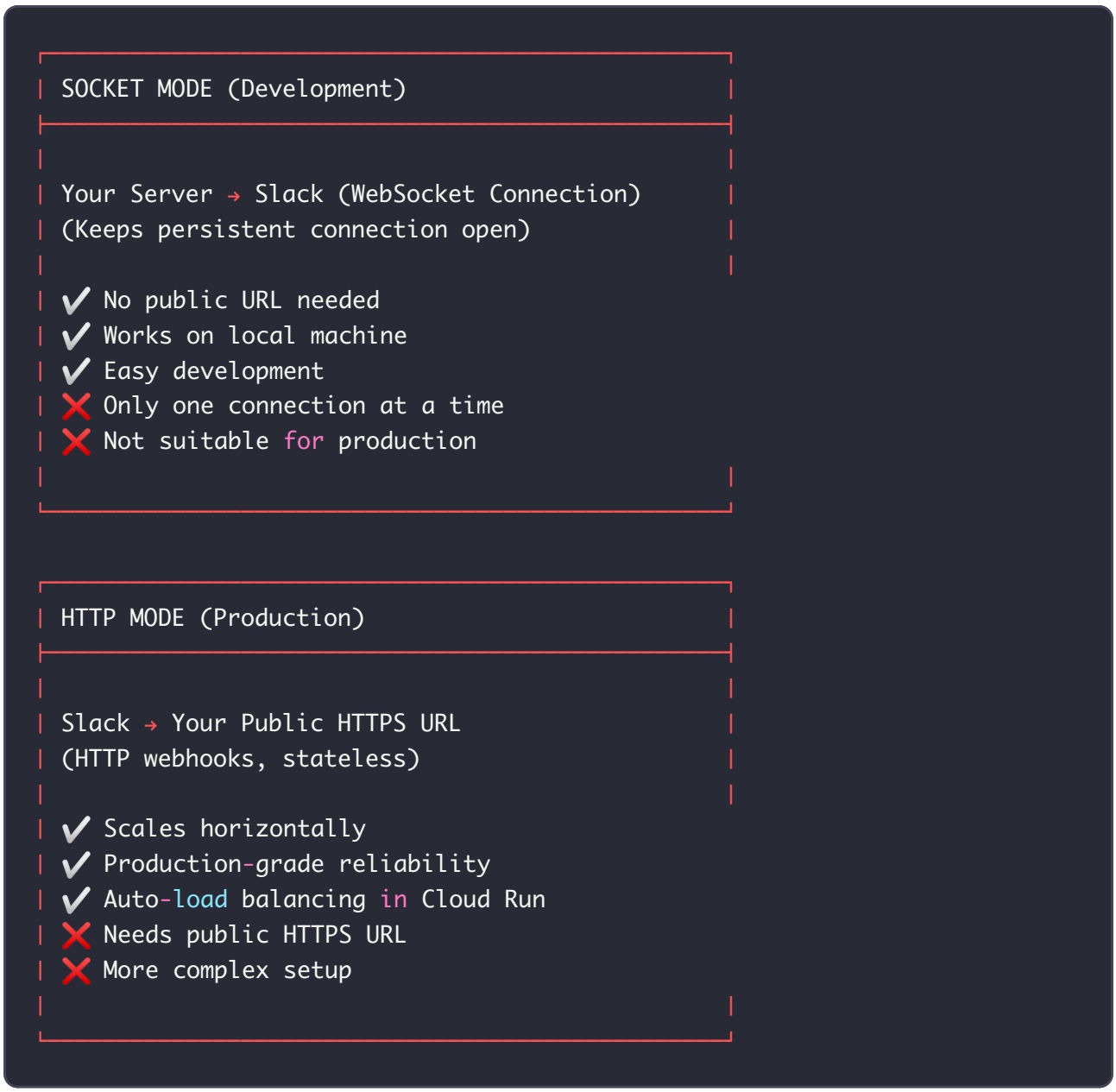
```
Layer 1: Slack Events  (Mentions, DMs, Reactions)
         ↓
Layer 2: Slack Bolt   (Routes to handlers, manages sessions)
         ↓
Layer 3: ADK Agent    (LLM, tool calling, decision logic)
         ↓
Layer 4: Tools        (Knowledge base, ticket system)
```

In this tutorial, you focus on Layers 2-4. We provide the Slack event handlers (Layer 1) as runnable code.

Key Mental Models

Mental Model 1: Socket Mode vs HTTP Mode

Understanding the **connection model** is crucial:



Decision Rule: Use Socket Mode while learning. Switch to HTTP Mode when deploying to production.

Mental Model 2: Agent Tool Execution

How does the ADK agent use your tools?

```
User: "What's the vacation policy?"
  ↓
Bot Handler (receives @mention)
  ↓
Sends text to ADK Agent
  ↓
Agent (with system prompt): "I should use search_knowledge_base"
  ↓
Calls: search_knowledge_base("vacation policy")
  ↓
Tool returns: {"status": "success", "article": {...}}
  ↓
Agent writes response: "Our PTO policy is 15 days per year..."
  ↓
Bot sends response back to Slack
```

Key insight: Tools return structured dicts with `status`, `report`, and data fields. The agent reads these and decides what to do next.

Mental Model 3: Session State Management

Conversation history needs to persist across messages:

```
Thread in Slack:
├─ User: "What's our password policy?"
|  Bot:   "Here's the password reset guide..."
|
├─ User: "How do I request a reset?"
|  Bot:   "You need to request via IT..."
|  (Bot remembers previous context!)
|
└─ User: "Create a ticket for me"
   Bot:   "Done! Ticket TKT-ABC created"
```

Implementation: Use `channel_id + thread_ts` as unique session key. Store session state in memory (development) or database (production).

Prerequisites & Setup

System Requirements

```
# Python 3.9 or later
python --version  # Should be >= 3.9

# pip (package manager)
pip --version
```

Required Accounts

1. Google AI API Key

Get from Google AI Studio (https://makersuite.google.com/app/apikey)

2. Slack Workspace

- Admin access to create apps
- Or create a test workspace at slack.com (https://slack.com/create)

Quick Start (15 Minutes)

:::tip Learning Approach

We provide a **working implementation** in `tutorial_implementation/tutorial33/` that you can run immediately, then study to understand how it works.

:::

Step 1: Get the Implementation

```
cd tutorial_implementation/tutorial33
pwd  # You should be in .../adk_training/tutorial_implementation/tutorial33
```

Step 2: Install and Test

```
make setup   # Install dependencies and package
make test    # Run 50 tests to verify everything works
```

Step 3: Configure Slack Tokens

Go to api.slack.com/apps (https://api.slack.com/apps) and create a new app:

1. **Click "Create New App" → "From scratch"**
2. **OAuth & Permissions**: Add these scopes:
3. `app_mentions:read` (receive @mentions)
4. `chat:write` (send messages)
5. `channels:history`, `groups:history`, `im:history` (read messages)
6. **Install to Workspace**: Get your **Bot Token** (starts with `xoxb-`)
7. **Socket Mode**: Enable it and create app-level token (starts with `xapp-`)

Save these tokens to `support_bot/.env`:

```
cp support_bot/.env.example support_bot/.env
# Edit support_bot/.env with your tokens
```

Step 4: Run the Bot

```
make slack-dev
```

You'll see: `✔️ Bot is running! Listening for mentions...`

Step 5: Test in Slack

Try these in any Slack channel or DM:

- `@Support Bot what's the vacation policy?`

- `@Support Bot how do I reset my password?`
- `@Support Bot I need to file an expense report`

The bot will:

1. Search the knowledge base 🔍
2. Find matching articles 📚
3. Respond with formatted answers ✔️

🎉 **You're done with Quick Start!**

Understanding the Architecture

Component Diagram

Socket Mode vs HTTP Mode

Aspect	Socket Mode	HTTP Mode
Connection	WebSocket (persistent)	HTTP webhooks
Setup	Easy (no public URL)	Requires public endpoint
Use Case	Development	Production
Latency	Low (~50ms)	Medium (~100ms)
Reliability	Reconnects automatically	Must handle retries
Deployment	Local or any server	Cloud Run, Heroku, etc.

Request Flow

1. User mentions bot: `@Support Bot how do I reset my password?`

2. Slack sends event to bot via Socket Mode/HTTP:

```
{
  "type": "app_mention",
  "user": "U12345",
  "text": "<@UBOT123> how do I reset my password?",
  "channel": "C67890",
  "ts": "1234567890.123456",
  "thread_ts": "1234567890.123456"
}
```

3. Bot handler processes event:

```python
@app.event("app_mention")
def handle_mention(event, say):
    # Extract message
    text = remove_mention(event["text"])
    thread_ts = event.get("thread_ts", event["ts"])

    # Get/create session for this thread
    session_id = f"{event['channel']}:{thread_ts}"
    session = get_or_create_session(session_id)

    # Send to ADK agent
    response = send_to_agent(session, text)

    # Reply in thread
    say(text=response, thread_ts=thread_ts)
```

4. ADK agent processes:

```
System: You are a support assistant...
User: how do I reset my password?
Agent: To reset your password:
1. Go to account.company.com
2. Click "Forgot Password"
3. Check your email...
```

5. Response sent back to Slack thread!

Building a Team Support Bot

Feature 1: Knowledge Base Search

Add a real knowledge base tool:

```
"""Enhanced bot with knowledge base search"""

from google.genai.types import Tool, FunctionDeclaration
import json

# Mock knowledge base (replace with real database/vector store)
KNOWLEDGE_BASE = {
    "password_reset": {
        "title": "How to Reset Your Password",
        "content": """To reset your password:
1. Visit https://account.company.com
2. Click "Forgot Password"
3. Enter your work email
4. Check your email for reset link
5. Create a new strong password (8+ chars, mix of letters/numbers/symbols)

If you don't receive the email within 5 minutes, check your spam folder or con
        "tags": ["password", "reset", "account", "login"]
    },
    "expense_report": {
        "title": "Filing Expense Reports",
        "content": """To file an expense report:
1. Log in to Expensify at https://expensify.company.com
2. Click "New Report"
3. Add expenses with receipts
4. Submit for manager approval
5. Reimbursement within 7 business days

Eligible expenses: Travel, meals (up to $50/day), software subscriptions (pre-

Questions? Email finance@company.com""",
        "tags": ["expense", "reimbursement", "finance", "expensify"]
    },
    "vacation_policy": {
        "title": "Vacation and PTO Policy",
        "content": """Our PTO policy:
• 15 days PTO per year (prorated for first year)
• 5 sick days per year
• 10 company holidays
• Unlimited unpaid time off (with manager approval)

To request time off:
1. Submit in BambooHR at https://bamboo.company.com
2. Get manager approval
3. Update your Slack status
4. Add to team calendar
```

```
Plan ahead for busy periods (Q4, product launches).""",
        "tags": ["vacation", "pto", "time off", "leave", "holiday"]
    },
    "remote_work": {
        "title": "Remote Work Policy",
        "content": """Remote work options:
• Hybrid: 3 days in office, 2 remote (standard)
• Full remote: Available for approved roles
• Temporary remote: For travel, emergencies (notify manager)

Requirements:
• Reliable internet (50+ Mbps)
• Quiet workspace
• Available during core hours (10am-3pm local time)
• Regular video presence in meetings

Equipment stipend: $500/year for home office setup.""",
        "tags": ["remote", "work from home", "hybrid", "wfh"]
    },
    "it_support": {
        "title": "IT Support Contacts",
        "content": """IT Support channels:
• Slack: #it-support (fastest, 9am-6pm ET)
• Email: it-help@company.com (24h response)
• Phone: 1-800-IT-HELPS (urgent issues only)
• Portal: https://support.company.com

Common issues:
• VPN: Use Cisco AnyConnect, credentials = AD login
• Printer: Add via System Preferences → Printers
• Software installs: Request in #it-support

Emergency (P0): Call phone number for system outages.""",
        "tags": ["IT", "support", "help", "technical", "vpn", "printer"]
    }
}

def search_knowledge_base(query: str) -> dict:
    """
    Search the company knowledge base.

    Args:
        query: Search query

    Returns:
        Dict with matching article or error
```

```
    """
    query_lower = query.lower()

    # Search by tags and content
    matches = []
    for key, article in KNOWLEDGE_BASE.items():
        score = 0

        # Check tags
        for tag in article["tags"]:
            if tag in query_lower:
                score += 2

        # Check title
        if any(word in article["title"].lower() for word in query_lower.split(
            score += 1

        # Check content
        if any(word in article["content"].lower() for word in query_lower.spli
            score += 0.5

        if score > 0:
            matches.append((score, article))

    if matches:
        # Return best match
        matches.sort(key=lambda x: x[0], reverse=True)
        best_article = matches[0][1]
        return {
            "found": True,
            "title": best_article["title"],
            "content": best_article["content"]
        }
    else:
        return {
            "found": False,
            "message": "I couldn't find a matching article. Try rephrasing or
        }

# Create agent with knowledge base tool
from google.adk.agents import Agent

agent = Agent(
    model="gemini-2.0-flash-exp",
    name="support_bot",
    instruction="""You are a helpful team support assistant.
```

```
Your responsibilities:
- Answer questions using the knowledge base
- Help with company policies and procedures
- Provide IT support guidance
- Be friendly, concise, and professional

Guidelines:
- ALWAYS use search_knowledge_base tool when users ask about:
  * Company policies (PTO, remote work, expenses)
  * IT support (passwords, VPN, printer, software)
  * Procedures and processes
- Format responses clearly with bullet points
- Include relevant links from knowledge base
- Use Slack formatting (*bold*, `code`, > quotes)
- If you can't find info, admit it and suggest contacting the right team

Remember: You're helping employees be productive!""",
    tools=[
        Tool(
            function_declarations=[
                FunctionDeclaration(
                    name="search_knowledge_base",
                    description="Search the company knowledge base for policie
                    parameters={
                        "type": "object",
                        "properties": {
                            "query": {
                                "type": "string",
                                "description": "Search query describing what t
                            }
                        },
                        "required": ["query"]
                    }
                )
            ]
        )
    ],
    tool_config={
        "function_calling_config": {
            "mode": "AUTO"
        }
    }
)

# Tool execution mapping
TOOLS = {
    "search_knowledge_base": search_knowledge_base
```

```
}

# Update handler to execute tools
@app.event("app_mention")
def handle_mention(event, say, logger):
    """Handle @mentions with tool calling."""
    try:
        user = event["user"]
        text = event["text"]
        channel = event["channel"]
        thread_ts = event.get("thread_ts", event["ts"])

        # Remove mention
        text = re.sub(r'<@[A-Z0-9]+>', '', text).strip()

        if not text:
            say(text="Hi! How can I help you?", thread_ts=thread_ts)
            return

        # Call agent directly - ADK handles tool execution automatically
        # Agent maintains conversation context and executes tools as needed
        full_response = agent(text)

        # Format and send
        formatted_response = format_slack_message(full_response)
        say(text=formatted_response, thread_ts=thread_ts)

    except Exception as e:
        logger.error(f"Error: {e}")
        say(text="Sorry, I encountered an error!", thread_ts=thread_ts)
```

Test it:

`@Support Bot how do I reset my password?`

Bot will search the knowledge base and provide the full password reset guide! 🔍

Feature 2: Rich Slack Blocks

Use Slack's Block Kit for beautiful messages:

```
def create_article_blocks(title: str, content: str) -> list:
    """Create rich Slack blocks for knowledge base article."""
    return [
        {
            "type": "header",
            "text": {
                "type": "plain_text",
                "text": f"📚 {title}",
                "emoji": True
            }
        },
        {
            "type": "divider"
        },
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": content
            }
        },
        {
            "type": "context",
            "elements": [
                {
                    "type": "mrkdwn",
                    "text": "💡 Need more help? Contact support@company.com"
                }
            ]
        }
    ]

def create_action_blocks(message: str, actions: list) -> list:
    """Create blocks with action buttons."""
    blocks = [
        {
            "type": "section",
            "text": {
                "type": "mrkdwn",
                "text": message
            }
        }
    ]

    if actions:
        blocks.append({
```

```
            "type": "actions",
            "elements": [
                {
                    "type": "button",
                    "text": {
                        "type": "plain_text",
                        "text": action["label"],
                        "emoji": True
                    },
                    "value": action["value"],
                    "action_id": action["action_id"]
                }
                for action in actions
            ]
        })

    return blocks

# Enhanced knowledge base search with blocks
def search_knowledge_base_with_blocks(query: str) -> dict:
    """Search and return formatted Slack blocks."""
    result = search_knowledge_base(query)

    if result["found"]:
        return {
            "found": True,
            "blocks": create_article_blocks(
                result["title"],
                result["content"]
            )
        }
    else:
        return {
            "found": False,
            "blocks": create_action_blocks(
                result["message"],
                actions=[
                    {
                        "label": "📧 Email Support",
                        "value": "email_support",
                        "action_id": "email_support"
                    },
                    {
                        "label": "💬 Open Ticket",
                        "value": "open_ticket",
                        "action_id": "open_ticket"
                    }
```

```
                ]
            )
        }

# Update handler to use blocks
@app.event("app_mention")
def handle_mention(event, say, client, logger):
    """Handle mentions with rich blocks."""
    # ... (same extraction logic)

    # After getting response from agent
    # Check if knowledge base was used
    if "search_knowledge_base" in full_response:  # Simplified check
        # Extract query from response
        # Call search_knowledge_base_with_blocks
        # Send blocks instead of plain text

        result = search_knowledge_base_with_blocks(text)

        if result["found"]:
            say(
                blocks=result["blocks"],
                thread_ts=thread_ts
            )
        else:
            say(
                blocks=result["blocks"],
                thread_ts=thread_ts
            )
    else:
        # Regular text response
        say(text=formatted_response, thread_ts=thread_ts)

# Handle button clicks
@app.action("email_support")
def handle_email_support(ack, body, say):
    """Handle email support button click."""
    ack()

    say(
        text="📧 You can email our support team at support@company.com\n\n" +
             "We typically respond within 24 hours on business days.",
        thread_ts=body["message"]["ts"]
    )

@app.action("open_ticket")
def handle_open_ticket(ack, body, say):
```

```
    """Handle open ticket button click."""
    ack()

    # Show modal for ticket creation
    client.views_open(
        trigger_id=body["trigger_id"],
        view={
            "type": "modal",
            "callback_id": "ticket_modal",
            "title": {
                "type": "plain_text",
                "text": "Create Support Ticket"
            },
            "submit": {
                "type": "plain_text",
                "text": "Submit"
            },
            "blocks": [
                {
                    "type": "input",
                    "block_id": "subject",
                    "label": {
                        "type": "plain_text",
                        "text": "Subject"
                    },
                    "element": {
                        "type": "plain_text_input",
                        "action_id": "subject_input"
                    }
                },
                {
                    "type": "input",
                    "block_id": "description",
                    "label": {
                        "type": "plain_text",
                        "text": "Description"
                    },
                    "element": {
                        "type": "plain_text_input",
                        "action_id": "description_input",
                        "multiline": True
                    }
                },
                {
                    "type": "input",
                    "block_id": "priority",
                    "label": {
```

```
                        "type": "plain_text",
                        "text": "Priority"
                    },
                    "element": {
                        "type": "static_select",
                        "action_id": "priority_select",
                        "options": [
                            {
                                "text": {"type": "plain_text", "text": "Low"},
                                "value": "low"
                            },
                            {
                                "text": {"type": "plain_text", "text": "Normal
                                "value": "normal"
                            },
                            {
                                "text": {"type": "plain_text", "text": "High"}
                                "value": "high"
                            },
                            {
                                "text": {"type": "plain_text", "text": "Urgent
                                "value": "urgent"
                            }
                        ]
                    }
                }
            ]
        }
    )
```

Now your bot sends **beautiful formatted messages** with buttons! 🎨

Feature 3: Create Support Tickets

Add ticket creation tool:

```
import uuid
from datetime import datetime

def create_support_ticket(subject: str, description: str, priority: str = "nor
    """
    Create a support ticket.

    Args:
        subject: Ticket subject
        description: Detailed description
        priority: Priority level (low, normal, high, urgent)

    Returns:
        Dict with ticket details
    """
    ticket_id = f"TKT-{uuid.uuid4().hex[:8].upper()}"

    # Mock ticket creation (replace with real ticketing system API)
    ticket = {
        "id": ticket_id,
        "subject": subject,
        "description": description,
        "priority": priority,
        "status": "Open",
        "created_at": datetime.now().isoformat(),
        "url": f"https://support.company.com/tickets/{ticket_id}"
    }

    return ticket

# Add to agent tools
FunctionDeclaration(
    name="create_support_ticket",
    description="Create a support ticket for issues that need human attention"
    parameters={
        "type": "object",
        "properties": {
            "subject": {
                "type": "string",
                "description": "Brief subject line for the ticket"
            },
            "description": {
                "type": "string",
                "description": "Detailed description of the issue"
            },
            "priority": {
```

```
                "type": "string",
                "description": "Priority level",
                "enum": ["low", "normal", "high", "urgent"]
            }
        },
        "required": ["subject", "description"]
    }
)

# Update TOOLS mapping
TOOLS = {
    "search_knowledge_base": search_knowledge_base,
    "create_support_ticket": create_support_ticket
}

# Agent instruction update
instruction="""...

When creating tickets:
- Use create_support_ticket for complex issues
- Set priority based on urgency
- Summarize the issue clearly
- Confirm ticket creation with user

..."""
```

Test it:

`@Support Bot my laptop won't connect to VPN, tried everything`

Bot creates a ticket and responds:

> I've created ticket **TKT-A1B2C3D4** for your VPN issue. Our IT team will reach out within 4 hours.
>
> Track it here: https://support.company.com/tickets/TKT-A1B2C3D4

🎟️ Ticket created!

Advanced Features

Feature 1: Context from Slack

Enrich agent with Slack context:

```
def get_user_info(user_id: str, client) -> dict:
    """Get user information from Slack."""
    try:
        response = client.users_info(user=user_id)
        user = response["user"]

        return {
            "name": user["real_name"],
            "email": user["profile"].get("email"),
            "title": user["profile"].get("title"),
            "team": user["profile"].get("team")
        }
    except Exception:
        return {}

def get_channel_info(channel_id: str, client) -> dict:
    """Get channel information."""
    try:
        response = client.conversations_info(channel=channel_id)
        channel = response["channel"]

        return {
            "name": channel["name"],
            "topic": channel.get("topic", {}).get("value"),
            "purpose": channel.get("purpose", {}).get("value")
        }
    except Exception:
        return {}

# Enhanced handler with context
@app.event("app_mention")
def handle_mention(event, say, client, logger):
    """Handle mentions with rich context."""
    # Get Slack context
    user_info = get_user_info(event["user"], client)
    channel_info = get_channel_info(event["channel"], client)

    # Add context to agent message
    context = f"""User context:
- Name: {user_info.get('name', 'Unknown')}
- Email: {user_info.get('email', 'Unknown')}
- Title: {user_info.get('title', 'Unknown')}

Channel context:
- Channel: #{channel_info.get('name', 'Unknown')}
- Topic: {channel_info.get('topic', 'N/A')}
```

```
User question: {text}"""

    # Send to agent with context - ADK Agent handles execution
    response = agent(context)

    # ... process response
```

Agent now knows who's asking and where! 🎯

Feature 2: Scheduled Messages

Send proactive reminders:

```
import schedule
import time
from threading import Thread

def send_daily_tip():
    """Send daily productivity tip to #general."""
    tips = [
        "💡 Tip: Use /support command for quick help without @mentioning me!",
        "📚 New knowledge base article: Check out our updated remote work poli
        "⏰ Reminder: Submit your timesheets before end of day Friday!",
        "🎉 Feature update: I can now create support tickets directly from Sla
    ]

    import random
    tip = random.choice(tips)

    app.client.chat_postMessage(
        channel="#general",
        text=tip
    )

# Schedule daily tips
schedule.every().day.at("10:00").do(send_daily_tip)

def run_schedule():
    """Run scheduled tasks in background thread."""
    while True:
        schedule.run_pending()
        time.sleep(60)

# Start scheduler
scheduler_thread = Thread(target=run_schedule, daemon=True)
scheduler_thread.start()
```

Feature 3: Analytics & Logging

Track bot usage:

```
import logging
from collections import defaultdict
from datetime import datetime

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler("bot.log"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

# Usage statistics
stats = defaultdict(int)

@app.event("app_mention")
def handle_mention(event, say, client, logger_obj):
    """Handle mentions with analytics."""
    # Log event
    logger.info(f"Mention from user {event['user']} in channel {event['channel

    # Track stats
    stats["mentions"] += 1
    stats[f"user_{event['user']}"] += 1
    stats[f"channel_{event['channel']}"] += 1

    # ... process mention

    # Log response
    logger.info(f"Responded with {len(full_response)} characters")
    stats["responses"] += 1

# Stats command
@app.command("/support-stats")
def handle_stats_command(ack, say, command):
    """Show bot usage statistics."""
    ack()

    # Admin only
    if command["user_id"] not in ADMIN_USERS:
        say("Sorry, this command is for admins only!")
        return
```

```
    message = f"""📊 *Support Bot Statistics*

Total mentions: {stats['mentions']}
Total responses: {stats['responses']}
Active users: {len([k for k in stats.keys() if k.startswith('user_')])}
Active channels: {len([k for k in stats.keys() if k.startswith('channel_')])}

Top users:
{get_top_users(stats, 5)}

Top channels:
{get_top_channels(stats, 5)}
"""

    say(text=message)

def get_top_users(stats, n=5):
    """Get top N users by interaction count."""
    user_stats = {k: v for k, v in stats.items() if k.startswith("user_")}
    sorted_users = sorted(user_stats.items(), key=lambda x: x[1], reverse=True

    return "\n".join([
        f"{i+1}. <@{user.replace('user_', '')}> - {count} interactions"
        for i, (user, count) in enumerate(sorted_users)
    ])
```

Production Deployment

Option 1: HTTP Mode (Recommended for Production)

Step 1: Update Bot for HTTP Mode

```
"""Production bot with HTTP mode"""

import os
from slack_bolt import App
from slack_bolt.adapter.flask import SlackRequestHandler
from flask import Flask, request

# Initialize Slack app (no Socket Mode)
app = App(
    token=os.environ.get("SLACK_BOT_TOKEN"),
    signing_secret=os.environ.get("SLACK_SIGNING_SECRET")
)

# ... (all your handlers)

# Flask app for HTTP endpoint
flask_app = Flask(__name__)
handler = SlackRequestHandler(app)

@flask_app.route("/slack/events", methods=["POST"])
def slack_events():
    """Handle Slack events via HTTP."""
    return handler.handle(request)

@flask_app.route("/health", methods=["GET"])
def health():
    """Health check endpoint."""
    return {"status": "healthy"}, 200

# Run Flask server
if __name__ == "__main__":
    port = int(os.environ.get("PORT", 8080))
    flask_app.run(host="0.0.0.0", port=port)
```

Step 2: Update Slack App Configuration

1. Go to **Event Subscriptions** in Slack app settings
2. Enable Events
3. Set Request URL: `https://your-app.run.app/slack/events`
4. Slack will verify the URL (make sure bot is running!)
5. Subscribe to bot events (same as before)

Step 3: Deploy to Cloud Run

Create `requirements.txt`:

```
slack-bolt==1.20.0
google-genai==1.41.0
python-dotenv==1.0.0
Flask==3.0.0
schedule==1.2.0
```

Create `Dockerfile`:

```
FROM python:3.11-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy bot code
COPY bot.py .

# Expose port
EXPOSE 8080

# Health check
HEALTHCHECK CMD curl --fail http://localhost:8080/health || exit 1

# Run bot
CMD ["python", "bot.py"]
```

Deploy:

```
# Deploy to Cloud Run
gcloud run deploy support-bot \
  --source=. \
  --region=us-central1 \
  --allow-unauthenticated \
  --set-env-vars="SLACK_BOT_TOKEN=xoxb-...,SLACK_SIGNING_SECRET=...,GOOGLE_API

# Output:
# Service URL: https://support-bot-abc123.run.app
```

Step 4: Update Slack Event URL

Go back to Slack app settings → Event Subscriptions → Update URL:

`https://support-bot-abc123.run.app/slack/events`

✔️ **Production bot is live!**

Production Best Practices

1. Rate Limiting

```python
from collections import defaultdict
import time

class RateLimiter:
    def __init__(self, max_requests=20, window=60):
        self.max_requests = max_requests
        self.window = window
        self.requests = defaultdict(list)

    def is_allowed(self, user_id):
        now = time.time()
        self.requests[user_id] = [
            req_time for req_time in self.requests[user_id]
            if now - req_time < self.window
        ]

        if len(self.requests[user_id]) < self.max_requests:
            self.requests[user_id].append(now)
            return True
        return False

rate_limiter = RateLimiter()

@app.event("app_mention")
def handle_mention(event, say):
    user_id = event["user"]

    if not rate_limiter.is_allowed(user_id):
        say(
            text="⚠️ You're sending too many requests. Please wait a minute!",
            thread_ts=event.get("thread_ts", event["ts"])
        )
        return

    # ... process normally
```

2. Error Recovery

```
from functools import wraps
import traceback

def retry_on_error(max_retries=3):
    """Retry decorator for Slack API calls."""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            for attempt in range(max_retries):
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    logger.error(f"Attempt {attempt + 1} failed: {e}")
                    if attempt == max_retries - 1:
                        raise
                    time.sleep(2 ** attempt)  # Exponential backoff
            return wrapper
        return decorator

@retry_on_error(max_retries=3)
def send_message_with_retry(channel, text, thread_ts):
    """Send message with automatic retry."""
    app.client.chat_postMessage(
        channel=channel,
        text=text,
        thread_ts=thread_ts
    )
```

3. Monitoring

```
from google.cloud import monitoring_v3

def log_metric(metric_name, value):
    """Log to Google Cloud Monitoring."""
    if os.getenv("ENVIRONMENT") != "production":
        return

    client = monitoring_v3.MetricServiceClient()
    project_name = f"projects/{os.getenv('GCP_PROJECT')}"

    series = monitoring_v3.TimeSeries()
    series.metric.type = f"custom.googleapis.com/slack_bot/{metric_name}"

    # ... (same as previous tutorials)

    client.create_time_series(name=project_name, time_series=[series])

@app.event("app_mention")
def handle_mention(event, say):
    start_time = time.time()

    # ... process mention

    latency = time.time() - start_time
    log_metric("response_latency", latency)
    log_metric("mentions", 1)
```

4. Session Cleanup

```python
from datetime import datetime, timedelta

# Clean up old sessions periodically
def cleanup_old_sessions():
    """Remove sessions older than 24 hours."""
    cutoff = datetime.now() - timedelta(hours=24)

    sessions_to_remove = []
    for session_id, session_data in sessions.items():
        if session_data.get("created_at", datetime.now()) < cutoff:
            sessions_to_remove.append(session_id)

    for session_id in sessions_to_remove:
        del sessions[session_id]
        logger.info(f"Cleaned up session: {session_id}")

# Run cleanup every hour
schedule.every().hour.do(cleanup_old_sessions)
```

Troubleshooting

Common Issues

Issue 1: Bot Not Responding

Symptoms:

- Mention bot, no response
- No errors in logs

Solutions:

```
# Check bot is running
curl https://your-bot.run.app/health

# Check Slack app config
# Event Subscriptions → Request URL should be verified (✓)

# Check bot token scopes
# OAuth & Permissions → Verify all scopes are added

# Check event subscriptions
# Event Subscriptions → Verify app_mention, message.im are subscribed
```

Issue 2: "Verification Failed" Error

Symptoms:

- Slack says request URL verification failed
- Events not reaching bot

Solution:

```
# Make sure bot handles challenge request
@flask_app.route("/slack/events", methods=["POST"])
def slack_events():
    # Slack sends challenge on initial setup
    if request.json and "challenge" in request.json:
        return {"challenge": request.json["challenge"]}

    # Normal event handling
    return handler.handle(request)
```

Issue 3: Rate Limit Errors

Symptoms:

- `ratelimited` error from Slack API
- Bot stops responding

Solution:

```python
from slack_sdk.errors import SlackApiError
import time

def send_message_safely(channel, text, thread_ts=None):
    """Send message with rate limit handling."""
    max_retries = 5

    for attempt in range(max_retries):
        try:
            app.client.chat_postMessage(
                channel=channel,
                text=text,
                thread_ts=thread_ts
            )
            return
        except SlackApiError as e:
            if e.response["error"] == "ratelimited":
                # Get retry-after header
                retry_after = int(e.response.headers.get("Retry-After", 1))
                logger.warning(f"Rate limited, waiting {retry_after}s")
                time.sleep(retry_after)
            else:
                raise
```

Issue 4: Tools Not Executing

Symptoms:

- Agent doesn't call functions
- Generic responses only

Solution:

```python
from google.adk.agents import Agent

# Verify tool registration - pass functions directly
agent = Agent(
    model="gemini-2.0-flash-exp",
    name="support_bot",
    instruction="...",
    tools=[search_knowledge_base, create_ticket]  # ✔️ Functions passed direct
)

# ADK automatically enables AUTO mode for function calling

# Verify tool mapping
TOOLS = {
    "search_knowledge_base": search_knowledge_base,  # ✔️ Name matches functio
    "searchKnowledgeBase": search_knowledge_base,    # ❌ Wrong name (use snak
}

# Check tool execution logic
for event in response_stream:
    if hasattr(event, 'function_calls'):  # ✔️ Check attribute exists
        for fc in event.function_calls:
            # Execute tool...
```

Issue 5: Session State Lost

Symptoms:

- Bot forgets conversation context
- Each message treated as new conversation

Solution:

```
# Use consistent session ID
def get_session_id(channel_id: str, thread_ts: str = None) -> str:
    """Generate consistent session ID."""
    # Use thread_ts for threaded conversations
    return f"{channel_id}:{thread_ts or 'main'}"

# Verify session is retrieved correctly
session_id = get_session_id(event["channel"], event.get("thread_ts"))

if session_id in sessions:
    session = sessions[session_id]  # ✔️ Reuse session
else:
    session = create_new_session()  # Create new
    sessions[session_id] = session

# Log for debugging
logger.info(f"Using session: {session_id}")
```

Common Pitfalls & How to Avoid Them

❌ Pitfall 1: Forgetting to Enable Event Subscriptions

The Problem:

You create the Slack app, install it, but bot never responds to @mentions.

Root Cause:

Events aren't subscribed in Slack app settings.

Solution:

```
Go to: OAuth & Permissions → Event Subscriptions
□ Enable Events
□ Subscribe to bot events:
  ✓ app_mention
  ✓ message.channels
  ✓ message.im
```

❌ Pitfall 2: Using Wrong Token for Socket Mode

The Problem:

```
Error: "invalid_auth"
```

Root Cause:
You used `SLACK_BOT_TOKEN` instead of `SLACK_APP_TOKEN` for Socket Mode.

Solution:
- Socket Mode needs `SLACK_APP_TOKEN` (starts with `xapp-`)
- HTTP webhooks need `SLACK_BOT_TOKEN` (starts with `xoxb-`)
- Both go in `.env` file

❌ Pitfall 3: Tool Functions Don't Match ADK Format

The Problem:

```
Agent: "I should call search_knowledge_base"
Result: ERROR - Tool not found
```

Root Cause:
Tool functions must return `{'status': 'success', 'report': '...'}` format.

Solution:

```
def my_tool(param: str) -> Dict[str, Any]:
    try:
        result = do_something(param)
        return {
            'status': 'success',
            'report': 'Human-readable message',
            'data': result  # Optional
        }
    except Exception as e:
        return {
            'status': 'error',
            'error': str(e),
            'report': 'Error message for user'
        }
```

❌ Pitfall 4: Session State Lost Between Messages

The Problem:

```
User: "What's the vacation policy?"
Bot:   "15 days PTO per year..."

User: "How do I request it?"
Bot:   "I don't know what you're asking about" 😟
```

Root Cause:
Each message creates a new session instead of reusing the thread session.

Solution:

```
# ✔️ Use thread_ts as part of session key
session_id = f"{channel_id}:{thread_ts}"

# Store conversation in persistent storage
if session_id not in sessions:
    sessions[session_id] = []

sessions[session_id].append({
    "role": "user",
    "content": message_text
})
```

❌ Pitfall 5: Agent Never Calls Tools

The Problem:

```
User: "Search for password policy"
Agent: "I don't have information about password policies"
```

Root Cause:
- Tools not properly registered
- System prompt doesn't encourage tool use
- Function names don't match tool names

Solution:

```
# ✔️ Register tools correctly
root_agent = Agent(
    name="support_bot",
    model="gemini-2.5-flash",
    tools=[
        search_knowledge_base,  # ✔️ Pass function directly
        create_support_ticket
    ]
)

# ✔️ Encourage tool use in instructions
instruction="""
When users ask about policies, use search_knowledge_base.
When they report issues, use create_support_ticket.
Always use tools when relevant!
"""
```

❌ Pitfall 6: Credentials Leaked in Code

The Problem:

```
SLACK_BOT_TOKEN = "xoxb-secret123"  # ❌ Don't do this!
```

Root Cause:
Hardcoding secrets in source code exposes them to git history.

Solution:

```
# ✔️ Always use environment variables
import os
from dotenv import load_dotenv

load_dotenv()
token = os.environ.get("SLACK_BOT_TOKEN")

# Add to .gitignore
echo ".env" >> .gitignore
```

✔️ Best Practice: Test Locally Before Deploying

```
# 1. Test in Socket Mode locally
make slack-dev

# 2. Run full test suite
make slack-test

# 3. Only then deploy to production
make slack-deploy
```

Next Steps

You've Mastered Slack + ADK! 🎉

You now know how to:

✔️ Build Slack bots with Google ADK
✔️ Handle mentions, DMs, and slash commands
✔️ Create rich Slack blocks and interactive buttons
✔️ Add knowledge base search and ticket creation
✔️ Deploy to production with HTTP mode
✔️ Implement rate limiting, monitoring, and error handling

Continue Learning

Tutorial 34: Google Cloud Pub/Sub + Event-Driven Agents
Build scalable, event-driven agent architectures

Tutorial 35: AG-UI Deep Dive - Building Custom Components
Master advanced CopilotKit features for web UIs

Tutorial 29: UI Integration Overview
Compare all integration approaches (Slack, Web, Streamlit, etc.)

Additional Resources

- Slack Bolt Documentation (https://slack.dev/bolt-python/)
- Slack Block Kit Builder (https://app.slack.com/block-kit-builder)
- ADK Documentation (https://google.github.io/adk-docs/)
- Slack API Reference (https://api.slack.com/methods)

🚀 Ready to Code?

View Working Implementation on GitHub → (https://github.com/raphaelmansuy/adk_training/tree/main/tutorial_implementation/tutorial33)

A complete, tested implementation is available with:
- ✔️ Root agent with tools exported
- ✔️ Knowledge base search tool (with 5 company knowledge articles)
- ✔️ Support ticket creation tool
- ✔️ 50 comprehensive tests (100% passing)
- ✔️ Slack Bolt Socket Mode integration ready
- ✔️ Production-ready structure with Cloud Run deployment

Quick Start:

```
cd tutorial_implementation/tutorial33
make setup  # Install dependencies and package
make test   # Run 50 tests
make dev    # Start ADK web interface at localhost:8000
```

Or clone and explore directly:

```
git clone https://github.com/raphaelmansuy/adk_training.git
cd adk_training/tutorial_implementation/tutorial33
make setup && make test
```

🎉 **Tutorial 33 Complete!**

Next: Tutorial 34: Google Cloud Pub/Sub Integration (./34_pubsub_adk_integration.md)

Questions or feedback? Open an issue on the ADK Training Repository (https://github.com/google/adk-training).

Generated on 2025-10-21 09:03:09 from 33_slack_adk_integration.md

Source: Google ADK Training Hub