BEST PRACTICE

Best Practices for Building AI Agents

Amazon Web Services (AWS)

1

Introduction

- Building intelligent agents that accurately understand and respond to user queries requires careful planning across multiple stages, from defining the agent's scope to creating a scalable infrastructure. AWS shares best practices for building generative AI applications with Amazon Bedrock Agents.
- Bedrock Agents leverage foundation models (FMs) to break down tasks, follow developer instructions to create orchestration plans, and use Retrieval-Augmented Generation (RAG) to access APIs and knowledge bases, providing accurate responses to user requests.

2

Collecting Ground Truth Data

- The foundation of a successful agent is high-quality ground truth data. The accurate, real-world observations used as reference for benchmarks to evaluate performance.
- Before building an agent, collect interactions or conversations that reflect expected behavior, including API usage, knowledge base access, and guardrails. This data enables effective testing and evaluation, identifies edge cases, and minimizes potential issues.
- To create a robust dataset, gather diverse examples covering various user intents and scenarios, with inputs and expected outputs for both simple and complex interactions. Update this dataset regularly based on user behavior, and use real customer interactions that are de-identified and anonymized.
- For instance, a banking assistant agent might use ground truth data that includes examples for APIs checking account balance and booking appointments, knowledge bases for banking advice and guardrails for blocking investment advice.
- It's also important to define information shared with the agent in production, such as session attributes, prompt session attributes, and knowledge base configurations.

Figure 1: Small subset of ground truth collected for a banking assistant agent

User Query	Session Attributes	Session prompt Attributes	Expected Response	API, Knowledge Bases and Guardrails invoked
What is my account balance?	None	None	Could you please provide the number of the account that you would like to check the balance for?	None
What is the balance for the account 1234?	user id 111	None	Your balance is X	Action Group: `check_account_balance(111, 1234)`

3

Defining Scope and Sample Interactions

- The next step is to define each agent's scope, outlining tasks it will handle and expected user interactions. This includes identifying primary functions, limitations, input formats, and output formats/styles.
- For example, a HR assistant agent's scope might include providing information on company HR policies but exclude handling sensitive employee data.
- Clearly defining the scope sets boundaries and expectations, guiding the development process and ensuring a focused, reliable AI agent.

Figure 2: High level HR assistant agent's scope

Primary functions:

– Provide information on company HR policies

– Assist with vacation requests and time-off management

– Answer basic payroll questions

Out of scope:

– Handling sensitive employee data

– Making hiring or firing decisions

– Providing legal advice

Expected inputs:

– Natural language queries about HR policies

– Requests for time-off or vacation information

– Basic payroll inquires

Desired outputs:

– Clear and concise responses to policy questions

– Step-by-step guidance for vacation requests

– Completion of tasks for book a new vacation, retrieve, edit and delete an existing request

– Referrals to appropriate HR personnel for complex issues

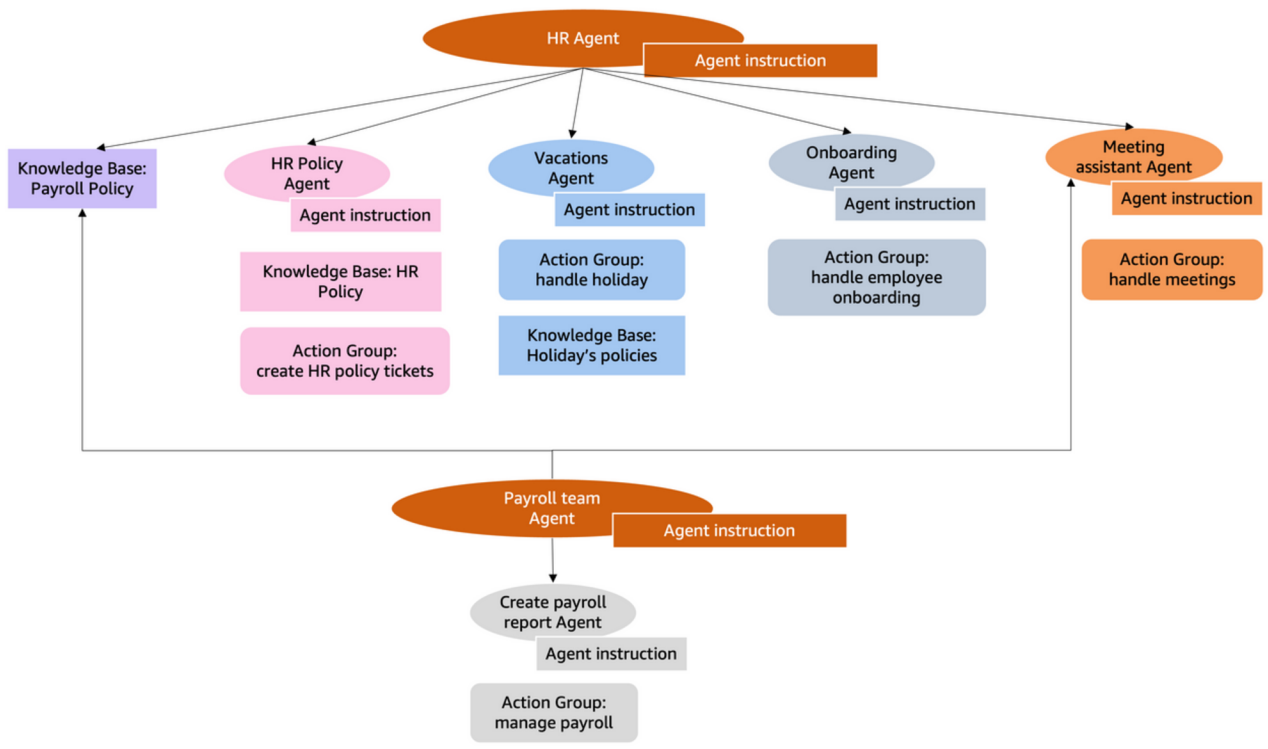
– Creation of an HR ticket for questions where the agent is not able to respond

4

Architecting Your Solution with Collaborative Agents

- In agent architecture, “divide and conquer” holds true. Building small, focused agents that interact improves modularity, maintainability, testing ease, and scalability. This approach also allows flexibility to use specific FMs for targeted tasks.
- For example, consider an HR assistant supporting internal employees and a payroll assistant for payroll team employees. Both handle some common tasks, like answering payroll questions and scheduling meetings, but with different scopes and permissions. The HR assistant accesses general company knowledge, while the payroll assistant accesses confidential data specific to payroll staff.
- In a single-agent setup, these functionalities overlap, requiring redundant action groups. But with multi-agent collaboration, each assistant relies on smaller, task-specific agents that serve distinct roles, like a meeting-scheduling agent shared across both assistants. Updates to shared functionality, like meeting scheduling, only need to be made once.

Figure 3: Multi-agent collaboration



5

Crafting the User Experience

- The personality of your agent shapes the user interaction. Planning the tone and greetings is key for a consistent and engaging experience. Key considerations include brand voice, target audience preferences, formality level, and cultural sensitivity.
- For example, a formal HR assistant should use titles and last names, maintaining a professional tone. In contrast, a friendly IT support agent can adopt a casual tone, addressing users by their first names and potentially using emojis or tech jokes to keep the conversation light.
- Ensure your agent's tone aligns with your brand identity and remains consistent across interactions. When multiple agents collaborate, set and enforce a unified tone across the application and its sub-agents.

Figure 4: Example prompt for a formal HR assistant

```
You are an HR AI Assistant, helping employees understand company policies and manage
their benefits. Always address users formally, using titles (Mr., Ms., Dr., etc.) and last names.
Maintain a professional and courteous tone throughout the conversation.
```

6

Clear Instructions and Definitions

- Clear communication is key for effective AI agents. Use unambiguous language in instructions, functions, and knowledge base interactions to avoid misinterpretation.
- Opt for simple, direct language and provide specific examples for complex concepts. Clearly define boundaries between similar functions and implement confirmation mechanisms for critical actions.

Figure 5: Example of a clear instructions

```
1. Verify the user's available time-off balance using the `checkTimeOffBalance`
2. If the requested time off is available, use the `bookTimeOff` function to res
3. If the time off is not available, inform the user and suggest alternative dat
4. Always confirm with the user before finalizing any time-off bookings.
```

Figure 6: Example of a clear function definitions

Function: get_user_address

Description: get the address of an user, including the city, country and zip code

Parameters:
Name: user_id
Description: 10 digits string starting with USR
Type: string
Required: True

Function: get_user_name

Description: get the first and last name of a user

Parameters:
Name: user_id
Description: 10 digits string starting with USR
Type: string
Required: True

7

Integrating Knowledge Bases

- Integrate your agents with existing enterprise knowledge bases to provide them with comprehensive information, enabling more accurate and context-aware responses. Access to up-to-date data improves response relevance, cites authoritative sources, and reduces the need for frequent model updates.

- Follow these steps when integrating a knowledge base with Amazon Bedrock:

  - **Index Documents:** Use Amazon Bedrock Knowledge Bases to index your documents in a vector database.
  - **Configure Access:** Set up your agent to access the knowledge base during interactions.
  - **Implement Citations:** Enable citation mechanisms to reference source documents in responses.
  - **Regular Updates:** Ensure consistent access to current information by implementing event-based synchronization of data sources using the StartIngestionJob API and an Amazon EventBridge rule, triggered periodically or by updates in your Amazon S3 bucket.

- Integrating Amazon Bedrock Knowledge Bases will enhance your application with semantic search capabilities. Use the knowledgeBaseConfigurations field in your agent's SessionState during the InvokeAgent request to manage interactions with your knowledge base, including setting result limits and filters.

8

Establishing Evaluation Criteria

- To effectively measure your AI agent's performance, define specific evaluation criteria. These metrics will help assess performance, identify improvement areas, and track progress.

- Key Evaluation Metrics:
  - **Response Accuracy:** Measures how well responses match ground truth data, indicating correctness and overall quality.
  - **Task Completion Rate:** Tracks the percentage of user interactions where the agent successfully completes tasks and meets user intent.
  - **Latency or Response Time:** Measures the time taken to respond after receiving input. Consider setting intermediate metrics to identify steps that need optimization.
  - **Conversation Efficiency:** Assesses how effectively the agent gathers necessary information during interactions.
  - **Engagement:** Evaluates the agent's ability to understand user intent, provide relevant responses, and maintain a conversational flow.
  - **Conversation Coherence:** Checks the logical progression and continuity of responses, ensuring context and relevance throughout the session.

- Define custom metrics tailored to your specific use case. For example, in an HR context, a relevant metric could be the number of tickets created when the agent cannot provide an answer.
- To establish a robust evaluation process:

  - Create a comprehensive test dataset based on ground truth data.
  - Develop automated scripts for measuring quantitative metrics.
  - Implement A/B testing to compare different agent versions or configurations.
  - Schedule regular human evaluations for qualitative factors.

9

Using Human Evaluation

- While automated metrics are valuable, human evaluation is also key for assessing and enhancing your AI agent's performance. Human evaluators offer nuanced feedback on areas that are hard to quantify, such as natural language understanding, response appropriateness, potential biases, and overall user experience.

- Best Practices for Human Evaluation:

  - **Diverse Evaluator Panel:** Assemble evaluators with varied perspectives.
  - **Clear Guidelines and Rubrics:** Develop straightforward evaluation criteria.
  - **Mix of Evaluators:** Include both expert evaluators (subject matter experts) and representative end-users.
  - **Collect Mixed Feedback:** Gather quantitative ratings alongside qualitative insights.
  - **Regular Analysis:** Continuously analyze results to identify trends and improvement areas.

10

Continuous Improvement: Testing Iterating, and Refining

- Building an effective AI agent is an iterative process. After creating a working prototype, it's key to test extensively, gather feedback, and continuously refine performance. This involves:
  - Comprehensive Testing: Use your ground truth dataset, conduct real-world user testing with a beta group, analyze agent logs and conversation traces, and regularly update instructions, function definitions, and prompts.
  - Performance Comparison: Evaluate performance across different frameworks (FMs).

- Utilize AI to generate diverse test cases.

- One tool is the agent trace, which records the prompts used by the agent at each step. This provides insights into the agent's reasoning process. Enable tracing in your InvokeAgent call during testing and disable it once validated.

- After collecting your ground truth dataset, define evaluation criteria. For instance, create a test dataset to compare results from your agent with those obtained by directly querying the vacations database. Evaluate agent behavior through human assessment or automate it using frameworks like Agent Evaluation. If model invocation logging is enabled, Amazon Bedrock Agents will provide CloudWatch logs to validate behavior, debug unexpected outputs, and make necessary adjustments.

- Plan A/B testing for different agent behaviors, such as formal versus informal tones for an HR assistant. During initial deployments, offer different agent versions to smaller user groups and evaluate their performance. Amazon Bedrock Agents provide built-in versioning capabilities to facilitate this testing phase.

11

Enable Comprehensive Logging + Observability

- Implementing thorough logging and observability practices from the start of your agent development is essential for debugging, auditing, and troubleshooting.
- Steps for Comprehensive Logging:
    - **Enable Model Invocation Logging:** Use Amazon Bedrock to capture prompts and responses securely in your account.
    - **Utilize Traces:** Amazon Bedrock Agents provides traces that detail the orchestration steps, underlying prompts, knowledge base references, and code generated by the agent. These trace events are streamed in real time, allowing you to customize user experience cues and keep end-users informed about their request's progress.
    - **Log Traces for Troubleshooting:** Use your agent's traces to track performance and resolve issues.
- Monitoring in Production
    - When deploying agent applications to production, establish a monitoring workflow to continuously analyze your logs. You can create a custom solution or use an open-source option like Bedrock-ICYM.

12

Use Infrastructure As Code

- Implementing Infrastructure as Code (IaC) frameworks is essential for reliable and iterative deployment in your software development projects. This approach enables you to create repeatable, production-ready agents that can be easily reproduced, tested, and monitored.

- Key Practices:
  - **Choose Your IaC Framework:** Use AWS CloudFormation, AWS Cloud Development Kit (AWS CDK), or Terraform with Amazon Bedrock Agents. Start with AWS Agent Blueprints construct, which offers templates for common capabilities that can be deployed and updated with a single AWS CDK command.
  - **Define Action Groups:** When creating agents, specify function definitions as JSON objects or provide an API schema in OpenAPI format. If an OpenAPI schema already exists, start with it. Ensure functions have clear natural language descriptions to help the agent understand when to use each function. If you don't have a schema, use simple JSON function definitions. You can quickly create a default AWS Lambda function using the Amazon Bedrock console.
  - **Focus on Reusability:** As your agent development scales, emphasize reusing components, including predefined guardrails, knowledge bases, and action groups across multiple agents.
  - **Accelerate Development with Generative AI:** Leverage generative AI to enhance the creation and maintenance of your code. Use the invoke model functionality in Amazon Bedrock, Amazon Q Developer support, or an AWS PartyRock application to generate the necessary IaC for function definitions and Lambda connections based on your action group metadata.
  - **Create a Test Pipeline:** Regardless of the approach you choose, establish a test pipeline to validate and run your IaC, optimizing your agent solutions.

13

Use SessionState for Additional Agent Context

- Utilize SessionState to enhance your agent's context. You can pass information to your Lambda function through SessionAttribute and provide context for your prompt using SessionPromptAttribute.
    - **SessionAttribute:** Use this for data only relevant to the Lambda function, such as a user authentication token.
    - **SessionPromptAttribute:** Use this for information that the large language model (LLM) requires for reasoning, like the current date and timestamp. This allows the agent to deduce details such as how many days remain until the next payment or how long it has been since an order was placed.

14

Optimize Model Selection for Cost and Performance

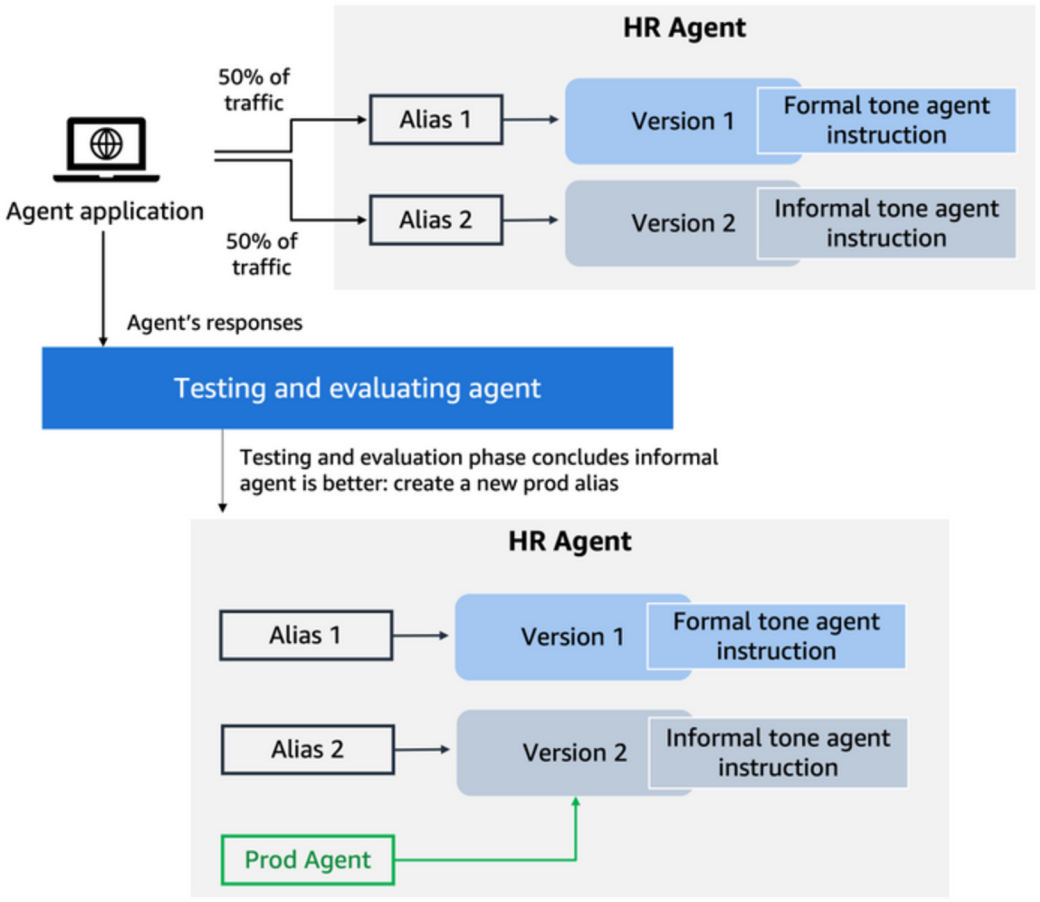
- Selecting the right foundation model (FM) for your agent is key. Experiment with different FMs to find the best fit for your application's cost, latency, and accuracy needs.
- Implement automated testing pipelines to gather evaluation metrics, which will help you make data-driven decisions. For simpler agents, consider using cost-effective models like Anthropic’s Claude 3 Haiku on Amazon Bedrock. For more complex applications, opt for advanced models like Claude 3.5 Sonnet or Claude 3 Opus.

15

Implement Robust Testing Frameworks

- Automating the evaluation of your agent can speed up development and ensure you deliver optimal solutions. Assess your agents on various dimensions, including cost, latency, and accuracy, using frameworks like Agent Evaluation to measure performance against predefined criteria.
- Utilize Amazon Bedrock's agent versioning and alias features to enable A/B testing during deployment. Define different behavioral aspects, such as a formal or informal tone for an HR assistant, and test these variations with subsets of your user group.
- This allows you to compare agent versions and refine their performance. The HR agent can then be updated post-evaluation, creating a new alias that points to the selected version for model invocation.

Figure 7: How the HR agent can be updated after a testing and evaluation phase



16

Use LLMs for Test Case Generation

- Leverage LLMs to generate test cases based on your agent's expected use cases. It's best to choose a different LLM for data generation than the one powering your agent. This approach accelerates the creation of comprehensive test suites, ensuring thorough coverage of potential scenarios.
- For example, you might use the following prompt to create test cases for an HR assistant agent that helps employees book holidays:

Figure 8: Prompt to create test cases for an HR assistant agent

```
Generate the conversation back and forward between an employee and an employee
assistant agent. The employee is trying to reserve time off.
The agent has access to functions for checking the available employee's time off,
booking and updating time off, and sending notifications that a new time off booking
has been completed. Here's a sample conversation between an employee and an employee
assistant agent for booking time off. Your conversation should have at least 3
interactions between the agent and the employee. The employee starts by saying hello.
```

17

Design Robust Confirmation and Security Mechanisms

- Implement strong confirmation mechanisms for critical actions in your agent's workflow. Clearly instruct the agent to seek user confirmation before executing functions that modify data or perform sensitive operations.
- This ensures reliable operation in production environments, moving beyond the proof-of-concept stage. For example, you might instruct your agent to confirm a vacation request before updating the user database:

Figure 9: Instruction for vacation request

```
You are an HR agent, helping employees … [other instructions removed for brevity]

Before creating, editing or deleting a time-off request, ask for user confirmation
for your actions. Include sufficient information with that ask to be clear about
the action that will be taken. DO NOT provide the function name itself but rather focus
on the actions being executed using natural language.
```

18

Implement Flexible Authorization and Encryption

- Use customer-managed keys to encrypt your agent's resources, and ensure that your AWS Identity and Access Management (IAM) permissions follow the principle of least privilege, granting access only to necessary resources and actions.
- When implementing action groups, leverage the sessionAttributes parameter in your sessionState to convey user roles and permissions for fine-grained access control.
- Additionally, use the knowledgeBaseConfigurations parameter in sessionState to set configurations for your knowledge base, such as filtering documents based on user group access through metadata.

19

Integrate
Responsible AI Practices

- Apply responsible AI practices to develop generative AI applications in an ethical, transparent, and accountable manner. Use Amazon Bedrock features to implement these practices at scale. For instance, employ Amazon Bedrock Guardrails to avoid sensitive topics, filter harmful content, and redact sensitive information to protect user privacy.

- Create organization-level guardrails for reuse across multiple applications, ensuring consistent responsible AI practices. Once established, associate guardrails with your agent using the built-in connection in Amazon Bedrock Agents

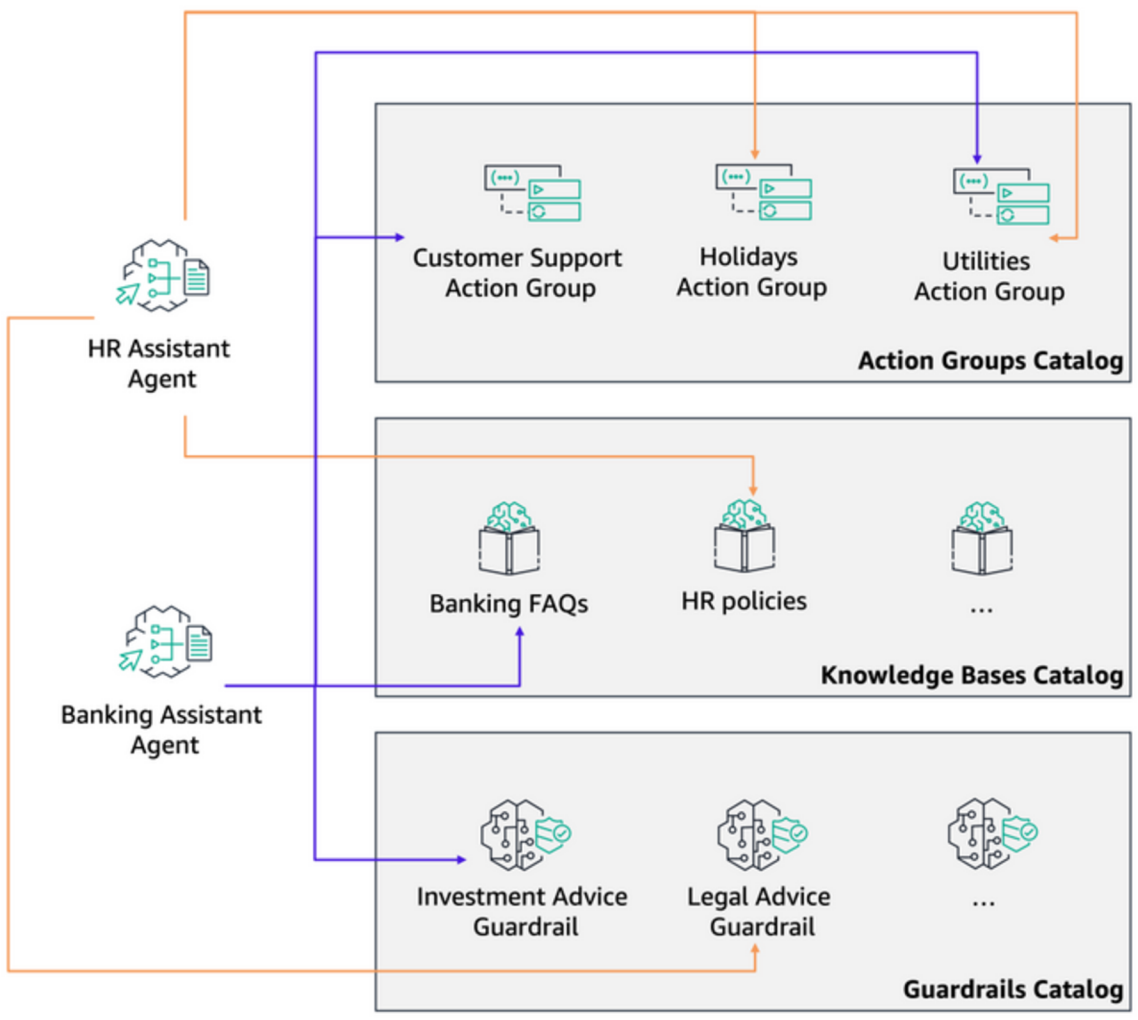
20

Build a Reusable Actions Catalog and Scale Gradually

- Once your first agent is successfully deployed, plan to reuse common functionalities like action groups, knowledge bases, and guardrails for other applications. Amazon Bedrock Agents allow you to create agents manually via the AWS Management Console, programmatically using SDKs, or through Infrastructure as Code (IaC) with CloudFormation templates, AWS CDK, or Terraform.

- For optimal reuse, deploy components using IaC to share them across applications. The figure below illustrates how a utilities action group can be reused by both an HR assistant and a banking assistant.

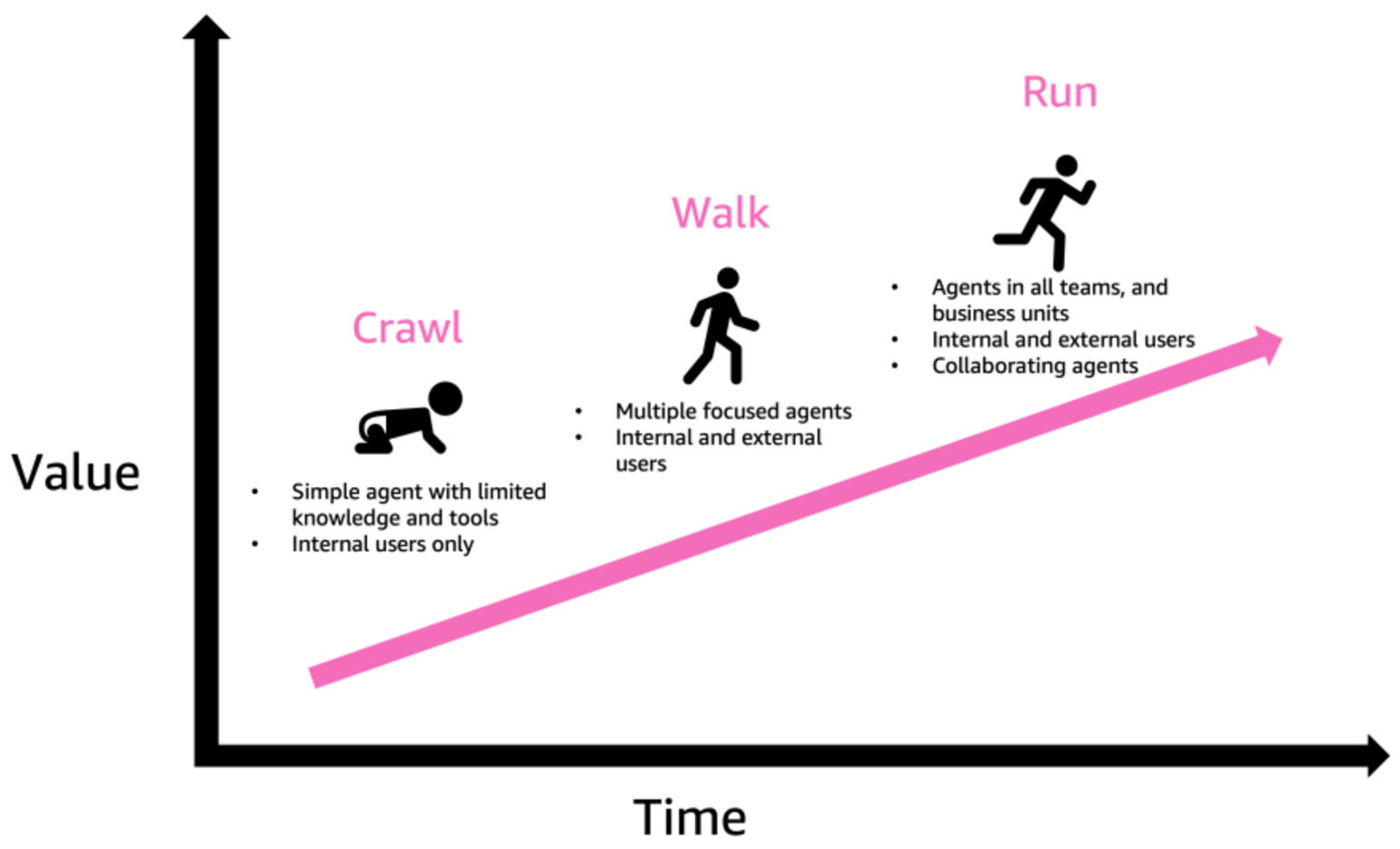
Figure 10: How a utilities action group can be reused



Follow a Crawl-Walk-Run Methodology for Scaling Agent Usage

- To effectively scale your agent usage, adopt the crawl-walk-run methodology. Begin with an internal application (crawl), then expand to a limited group of external users (walk), and finally roll out to all customers (run), and eventually incorporate multi-agent collaboration.
- This approach builds reliable agents for mission-critical operations while minimizing rollout risks. The figure below illustrates this process.

Figure 11: Crawl-Walk-Run methodology for scaling agent usage



22

Conclusion

- By following these best practices for architecture and development, you'll be equipped to build robust, scalable, and secure agents that effectively serve users and integrate seamlessly with existing systems.
- To get started, explore the Amazon Bedrock samples repository: https://github.com/aws-samples/amazon-bedrock-samples/tree/main/agents-and-function-calling/bedrock-agents

- This is a summary based on AWS's blog series 'Best practices for building robust generative AI applications with Amazon Bedrock Agents':
  - Part 1: https://aws.amazon.com/blogs/machine-learning/best-practices-for-building-robust-generative-ai-applications-with-amazon-bedrock-agents-part-1/
  - Part 2: https://aws.amazon.com/blogs/machine-learning/best-practices-for-building-robust-generative-ai-applications-with-amazon-bedrock-agents-part-2/
  - AWS Authors: Maira Ladeira Tanke, Mark Roy, Navneet Sabbineni, Monica Sunkara

Want to level up with Generative AI? Follow  Lewis Walker

Repost this to help your network