This paper explains the key building blocks to communications apps, and provides best practices on how the Twilio Runtime provides the tools, services, and elastic runtime to make developers more efficient throughout the development lifecycle — from building, to deploying, operating, and scaling solutions.
Goals of This Paper

This paper explains the key building blocks to communications apps, and provides best practices on how the Twilio Runtime provides the tools, services, and elastic runtime to make developers more efficient throughout the development lifecycle — from building, to deploying, operating, and scaling solutions.

What's inside

- Blueprint for Building Modern Communications Apps
  - The Rise of Communications Apps
  - Traditional App Architecture
  - Communications App Architecture
  - A Communications App Blueprint
- The Twilio Runtime
  - Features of the Twilio Runtime
  - Twilio Runtime: Extends the Power of the Twilio Customer Engagement Platform
The Rise of Communications Apps

The telephone allowed people thousands of miles away to communicate as if they were in the same room. The television let viewers glimpse exotic locations from the comfort of their living room. The internet offered us a platform to share the world’s knowledge. The technology that makes this possible is constantly evolving, yet the core premise has remained constant — to empower communications.

Communications apps center around digital interactions a user has with a company during every stage of a customer journey, whether it is browsing a product catalog, tracking a shipment, or resolving an issue with customer service. They are never an isolated solution. Each step of the customer journey is part of a broader conversation before the customer and the business. This is how communications apps differ from traditional app development. communications apps center around the digital interactions — the conversation.

Communications apps require more than traditional technology fundamentals: databases, platforms, micro-services, etc. and application-based input models: browser forms, app dialogs, interaction points. The new technology stack for communications developers must support the ability to implement conversational workflows to drive interactions that combine communication APIs, state management, connectivity to existing systems and processes, and custom logic to deliver an end-to-end experience, and eliminate the need to spend development cycles on operating and managing servers at scale. Further, with the rise of chatbots, and A.I., the need for developers to easily tap into these trends without being data scientists, is critical to delivering an app that satisfy the end user’s expectations.
Traditional App Architecture

Traditional app architectures for consumer-facing apps utilize a number of services from in-app logic, integration to off-the-shelf and custom backends, PaaS providers for deployment, and communication APIs from cPaaS services. Development teams are required to invest heavily in supporting multiple app development environment especially around conversational and business logic. Utilizing cloud application development platforms such as Heroku, Microsoft Azure, or Google Cloud can reduce the complexity of managing servers and infrastructure. However, for the specific needs of communications apps, developers are still required to stitch all of these disparate services together. The result is they spend significant effort working on plumbing, rather than building app functionality.

For example, in order to build a communications app for hotel concierge using traditional app architectures, developers must write integration and business logic, create custom state management, build or deploy a workflow engine, integrate with CDNs, connect it to a communications API provider, and spin up servers to manage everything.
Typically replies within an hour

**Messenger**

**Chatbot**

**Mobile App**

**Start conversation**

"I'd like to upgrade my room"

**Continue conversation**

"I'll send my details when I get in front of my laptop"

**Communications API provider**

**REST API**

**Manage conversation state**

**Agent or IVR**

"I see you are a preferred customer"

**Agent or IVR**

"Here are the available rooms. To upgrade please provide your credit card details"

**Integration Logic**

**Business Logic**

Calc Upgrade Fee

**Data Store**

**Asset & Media Storage**

**Payment Processing**

**Sync State Between Mobile & Web**

**Custom App Code**

**Email Confirmation**
Communications App Architecture

In order to alleviate much of the burden that traditional app architectures place on communications app developers, a new architecture approach is required. This architecture factors in new technology fundamentals, and delivers them in a single environment, with the goal of improving developer productivity.

Communications apps must be multi-channel. The ability to write logic and share it across devices and channels requires access to serviceless compute resources and conversational workflow tools, without the need to build your own, or manage servers or build your own, is a critical requirement. Further, once a communications app is live, developers need insights into the performance of their app. By eliminating the need for multiple environments to build and deliver an app, developers gain the additional benefit of centralized debugging and logging to improve operational efficiency.

Communications apps are transient by nature. Once a conversation between user and business has completed, information is typically stored within existing systems, such as help desks or customer profile stores. The communications app architecture must promote the ability to connect to these existing systems rather than requiring developers to create new data stores, whether they are on-premises, or cloud-based.

Lastly, as many organizations rely on agencies to develop digital experience applications, especially for consumer-facing applications, communications app architectures should be a natural complement to this development model by leveraging Continuous Integration strategies, version control, and the ability to pipe logs and alerts into existing organizational response management systems.

To demonstrate, the diagram below revisits the hotel concierge scenario, by depicting a best practice communications app architecture which leverages a development environment which provides native support for communications apps.
"I'd like to upgrade my room"

"I'll send my details when I get in front of my laptop"

"Here are the available rooms. Please complete the following credit card flow"

"Agent or IVR: I see you are a preferred customer"

"Agent or IVR: Here are the available rooms. Please complete the following credit card flow"

"Typically replies within an hour"
Communications App Blueprint

With an understanding of how communications app architectures are different, we can begin to look at what components are required to build these apps. Developers who build communications apps will be primarily responsible for creating the conversational workflows and associated business logic functions. However, as communications apps are typically an extension to consumer apps, in particular, mobile apps, the same developer often works on both aspects. The following blueprint provides insight into all major development aspects and best practices required to make communications apps successful.

**Consumer app development**

Communications apps interfaces vary considerably based on use case. These interfaces may include a mobile or messenger app, SMS, or even no user interface at all such as is the case with voice calling and VPAs. Communications app developers should consider leveraging a cPaaS provider to provide access to a broad set of channel APIs and conversational workflow logic. By extracting communication logic from within the app and delegating to the cloud, developers will benefit from shorter development cycles and improved application maintenance. Users will benefit by receiving messages on the channel and device they prefer.

**Authentication & identity**

Developers must take steps to ensure that customer identity and personal information is protected at all times. With the rise of identity theft and credit card fraud, governments are taking an active role in requiring certain actions to protect citizens. For example, the PSD2 initiative in Europe requires that every transaction over €30.00 be protected with two-factor authentication. Developers must support these new requirements by considering how they will provide additional protection for customer authentication and identity. They should look for vendors who offer either identity services, second-factor authentication APIs which can be called via apps, or provide support for separate apps that offer identity sync across devices and support for all major social and ecommerce platforms.
Access to communication channels

Communication channels may include SMS, voice, video, fax, chat, and messenger platforms. Rather than building custom code, developers should rely on cPaaS vendors to deliver these services via APIs and SDKs for all major programming languages. When selecting a cPaaS provider, developers should look for a platform with a breadth of channels so they reach users, and tailor messages to the right medium. Intuitive and well-documented APIs will also help developers build more swiftly, further improving developer productivity.

Stateful conversation workflow modeling

At the core of communications apps is the ability to visually model the conversation — the customer engagement touchpoints. When modeling conversations, developers should consider:

• The communications event that triggers the conversation: an SMS notification is sent indicating that a ride-share has arrived, an email that an order has shipped, a key is pressed in an IVR, the customer uses a messenger chatbot to request assistance via a customer service agent, and so on.

• The message was received, or responded to, via which channel.

• What contextual information will improve the customer experience: customer history from a CRM, for example.

• What state should be maintained during the conversation: eliminate the need for a customer to repeat themselves if the conversation is handed over to another agent, or the customer continues the conversation on another device.

• Where persistent data should be stored.
Serverless custom logic functions

Developers should minimize communications logic within application code and delegate to cloud-based runtimes as serverless functions. These functions should utilize modern programming languages like Node.js that are designed for real-time, multi-user applications. Developers should be able to discover and call logic functions directly from within a conversational workflow. These logic functions should be accessible via APIs and webhooks as part of a broader micro-service architecture to allow the communications app to be an extension to existing customer service strategies.

As a good rule-of-thumb, communications apps should manage transient conversational state data only. Developers should rely on cloud-based logic functions to either persist data to, or retrieve additional contextual information that enriches the conversation from, external data sources.

Custom logic serverless functions will typically fall into three primary categories:

- Business logic, such as calculating change fees for airline tickets.
- Inbound integration for contextual information, such as CRM for past hotel reservations and customer preferences, or HTTP endpoint.
- Outbound integration for data persistence or backend system connectivity such as sending an order to an ERP.

When evaluating the complexity of logic that should be included in custom functions that support outbound integrations, developers should consider the use of integration providers such as Mulesoft, and Jitterbit who provide robust connectivity to many backend systems, therefore eliminating the requirement for developers to build complex success-fail-retry logic.

Storage and efficient delivery of media assets

Customers expect rich media to be part of their conversation. Developers need the ability to store media assets at low cost and in a scalable environment without introducing an additional integration point. Media assets should leverage a content
delivery network (CDN) and efficient caching strategies utilized to reduce delivery and rendering time especially when network latency can adversely affect the customer experience.

**Conversational sync & omnichannel experiences**

Successful communications apps must provide a consistent omnichannel experience regardless of interface. For retail in particular, omnichannel shoppers have a 30% higher lifetime value than those who shop using only one channel. A consumer may browse and add items to a shopping cart on their mobile device during an afternoon commute, and check out on their laptop when they are at home later in the evening. Communications apps must be able to seamlessly sync state as a core requirement to providing excellent customer service through omnichannel experiences.

**Natural language understanding & task models**

By its nature, a conversation inherently involves two parties. Traditionally, both parties are humans, conversing via SMS, or Messenger app with a call center agent. But, with the rise of Artificial Intelligence, analysts are predicting that by 2021, 15% of all customer service interactions will be completely handled by AI. When creating communications apps, businesses must also consider how they incorporate digital participants such as chatbots.

Chatbots must understand natural language (NLU) to participate in a communications app. Take the following example of a customer asking a simple phrase like, “I need a hotel room in San Francisco from April 4th to 8th near the Convention Center.” NLU can be utilized to parse the phrase into conversational intent, context, and sentiment:

```plaintext
Need: hotel { intent }
- context: San Francisco { city }
- context: April 4th { check-in }
- context: April 8th { check-out }
Need: near the convention center { intent }
- sentiment: 0.5 { neutral }
```

---

Based on the identification of need and context, a bot may be able to complete a customer’s request, ask for additional information, or make the determination to add a human to the conversation. In the example above, the bot may determine it needs more information around intent and responds “There are four convention centers in San Francisco: Press 1 for Moscone Center, 2 for Masonic Center, 3 for San Francisco Conference Center, and 4 for Bill Graham Auditorium. Press 1 for Moscone, 2 for Masonic.”

Developers should consider the style of chatbot they need to build. Then, they can identify the correct technology to implement it:

- Can the bot interaction be designed with a static workflow model? If so, visual modeling tools will work best.

- Does the chatbot need to support dynamic modeling and complex decision logic? If so, a vendor such as wit.ai (Facebook), api.ai (Google), and viv.ai (Samsung), who provides deeper AI support should be considered.

- How broad will the machine learning model need to be? Watson (IBM) provides support for broadly applicable Machine Learning (ML) models (geographical information such as conference centers in San Francisco, weather, etc.), Einstein (Salesforce) supports custom ML models for your business, and Twilio provides strong support for visual modeling and NLU based on conversational task-based ML models (IVR, etc.).

**Augment existing customer service systems**

Successful communications apps should be an extension to, and augment existing customer service systems. In particular, consumers should be able to connect directly with customer service agents via preferred channels either in-app or as part of a messaging platform like WhatsApp. Consumers should be able to start a conversation with a customer service agent on one channel and continue on another channel without the need to repeat customer information or context. This omnichannel experience must span both the consumer and the agent through integration with contact centers without the need to introduce complex integration logic.
Developer productivity

Modern developer tools and processes are critical in keeping up with the demands of business and delivering apps and improving time to market. Communications apps are no different. They should allow developers to access serverless compute resources, use their preferred IDE, and utilize command line tools and shells to script deployments and operationalize runtime insights including real-time debugging, centralized logging, and customizable alerts.
The Twilio Runtime

The Twilio Runtime is a fully managed, elastic serverless runtime and suite of development tools that enables developers to rapidly build, deploy, operate, and manage communications apps and experiences, at scale. It combines the benefits of traditional serverless platforms: reductions in development cycles by eliminating the need to maintain servers, and worry about capacity planning, with the tools and services communications app developers need.

Features of the Twilio Runtime

Create Conversational Workflow with Drag & Drop
Build and run stateful workflows and access context variables with Studio, a rich multi-channel visual modeling tools for creating IVRs, chatbots, and more.

Instant Access to Serverless Compute
Reduce operating costs, and developer workload by never worrying about CDNs, capacity planning, or managing services. Store media assets and write custom serverless functions in the same elastic compute runtime environment as your communication APIs.

Painless Operation & Management of Your Apps
Quickly identify and resolve performance bottlenecks and errors with real-time debug insights, detailed logs, and customizable alert management.

Improve Developer Productivity
Provision phone numbers from the command line, automate typical development tasks like CI builds, and easily add communications capabilities in any app with SDKs for major languages.
Twilio Runtime: Extends the Power of the Twilio Customer Engagement Platform

Twilio Runtime is a key part of the Twilio Customer Engagement Platform. It leverages Twilio’s deep experience in communications to include everything a developer needs to build communications apps. With the rock-solid global Super Network infrastructure, you can rest easy knowing that your messages will be received in the right channel for your customer. Getting started is quick, simple, and free. Check out twilio.com/runtime for more information.
Thanks for reading.

Would you like to learn more about what Twilio can do for your business?

Talk to us