Types of Providers in Riverpod

Riverpod provides different types of providers for managing state efficiently in Flutter. Here's a comprehensive guide to all possible provider types in **Riverpod** with examples.

🚀 Types of Providers in Riverpod

1. **Provider (Read-Only State)**
2. **StateProvider (Mutable State)**
3. **FutureProvider (Asynchronous State)**
4. **StreamProvider (Stream-Based State)**
5. **NotifierProvider (State Notifier for Complex Logic)**
6. **AsyncNotifierProvider (Asynchronous Logic Handling)**
7. **ChangeNotifierProvider (Using ChangeNotifier)**
8. **Provider.autoDispose (Auto-Cleanup on Dispose)**

1️⃣ Provider (Read-Only State)

Used to expose **immutable values** (constants, configurations, computations).

```
dart
CopyEdit
import 'package:flutter_riverpod/flutter_riverpod.dart';

final greetingProvider = Provider<String>((ref) {
  return "Hello, Riverpod!";
});
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final greeting = ref.watch(greetingProvider);
    return Text(greeting);
  },
);
```

2️⃣ StateProvider (Mutable State)

Used when the state needs to be **modifiable** (like a counter).

```
dart
CopyEdit
final counterProvider = StateProvider<int>((ref) => 0);
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final counter = ref.watch(counterProvider);
    return Column(
      children: [
        Text('Counter: $counter'),
        ElevatedButton(
          onPressed: () {
            ref.read(counterProvider.notifier).state++;
          },
          child: Text('Increment'),
        ),
      ],
    );
  },
);
```

3️⃣ FutureProvider (Asynchronous State)

Used when fetching **data asynchronously**, such as an API call.

```
dart
CopyEdit
final userProvider = FutureProvider<String>((ref) async {
  await Future.delayed(Duration(seconds: 2));
  return "User: John Doe";
});
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final userAsync = ref.watch(userProvider);
    return userAsync.when(
      data: (data) => Text(data),
      loading: () => CircularProgressIndicator(),
      error: (err, stack) => Text("Error: $err"),
```

```
    );
  },
);
```

4️⃣ StreamProvider (Stream-Based State)

Used for **real-time data updates**, such as Firebase or WebSockets.

```
dart
CopyEdit
final timeStreamProvider = StreamProvider<DateTime>((ref) async* {
  while (true) {
    await Future.delayed(Duration(seconds: 1));
    yield DateTime.now();
  }
});
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final timeAsync = ref.watch(timeStreamProvider);
    return timeAsync.when(
      data: (time) => Text(time.toString()),
      loading: () => CircularProgressIndicator(),
      error: (err, stack) => Text("Error: $err"),
    );
  },
);
```

5️⃣ NotifierProvider (State Notifier for Complex Logic)

Best for **complex state logic and multiple state changes**.

```
dart
CopyEdit
class CounterNotifier extends Notifier<int> {
  @override
  int build() => 0;

  void increment() => state++;
  void decrement() => state--;
}

final counterNotifierProvider = NotifierProvider<CounterNotifier, int>(() => CounterNotifier());
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final counter = ref.watch(counterNotifierProvider);
    return Column(
      children: [
        Text('Counter: $counter'),
        ElevatedButton(
          onPressed: () => ref.read(counterNotifierProvider.notifier).increment(),
          child: Text('Increment'),
        ),
      ],
    );
  },
);
```

6️⃣ AsyncNotifierProvider (Asynchronous Logic Handling)

Used for handling **async operations inside Notifier**.

```
dart
CopyEdit
class UserNotifier extends AsyncNotifier<String> {
  @override
  Future<String> build() async {
    await Future.delayed(Duration(seconds: 2));
    return "User: John Doe";
  }
}

final userNotifierProvider = AsyncNotifierProvider<UserNotifier, String>(() => UserNotifier());
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final userAsync = ref.watch(userNotifierProvider);
    return userAsync.when(
      data: (data) => Text(data),
      loading: () => CircularProgressIndicator(),
      error: (err, stack) => Text("Error: $err"),
    );
  },
);
```

7️⃣ ChangeNotifierProvider (Using ChangeNotifier)

Useful when working with **ChangeNotifier-based state management**.

```
dart
CopyEdit
class CounterNotifier extends ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }
}

final counterNotifierProvider = ChangeNotifierProvider((ref) => CounterNotifier());
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final counterNotifier = ref.watch(counterNotifierProvider);
    return Column(
      children: [
        Text('Counter: ${counterNotifier.count}'),
        ElevatedButton(
          onPressed: () => counterNotifier.increment(),
          child: Text('Increment'),
        ),
      ],
    );
  },
);
```

8️⃣ Provider.autoDispose (Auto-Cleanup on Dispose)

Use `.autoDispose` when a provider should be **disposed automatically** when not needed.

```
dart
CopyEdit
final autoDisposeProvider = Provider.autoDispose<int>((ref) {
  return 42;
});
```

Usage in Widget:

```
dart
CopyEdit
Consumer(
  builder: (context, ref, child) {
    final number = ref.watch(autoDisposeProvider);
    return Text('AutoDisposed Number: $number');
  },
);
```

🌟 Summary Table

Provider Type	Usage
Provider	Read-only immutable state
StateProvider	Mutable state (simple variables)
FutureProvider	Async operations (API calls)
StreamProvider	Real-time streaming data
NotifierProvider	Complex state logic (multiple mutations)
AsyncNotifierProvider	Async logic inside state
ChangeNotifierProvider	Integrate ChangeNotifier
Provider.autoDispose	Automatically clean up state

🚀 Conclusion

- Use `Provider` for **immutable data**.
- Use `StateProvider` for **simple, mutable state**.
- Use `FutureProvider` and `StreamProvider` for **async data fetching**.
- Use `NotifierProvider` or `AsyncNotifierProvider` for **complex state logic**.
- Use `.autoDispose` when **cleanup is needed automatically**.